2020

Simulación de una red de colas en tandem

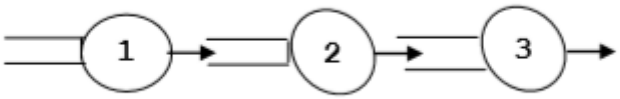
KHALID AL FOZAN DE LA CERDA, BO0264

TS | ETSISI | UPM

Contenido

Introducción

El trabajo final de la asignatura Técnicas de simulación ha sido realizar una simulación de la siguiente red en colas en tándem:



Las especificaciones de dicha res son las siguientes:

- El tiempo entre llegadas a la cola 1 es exponencial con tasa de llegada 2 λ = clientes/minuto. A las colas 2 y 3 no llegan clientes desde el exterior.
- El tiempo de servicio de cada uno de los 3 servidores es:
    - a) Exponencial de media 1/3 de minuto (μ = 3) para el servidor 1, 1/4 de minuto (μ = 4) para el servidor 2 y 1/6 de minuto (μ = 6) para el servidor 3.
    - b) Erlang de parámetro α = 3 (suma de 3 exponenciales). El tiempo medio de servicio es de 1/3 de minuto (con la notación habitual el parámetro β = 9) para el servidor 1, 1/4 de minuto (β = 12) para el servidor 2 y 1/6 de minuto (β = 18) para el servidor 3.
- La capacidad de las colas 1 y 2 es infinita, la de la 3 finita. Si hay 7 clientes en la tercera cola se pierden los clientes que lleguen.
- Los clientes que salen del servidor 1 van al servidor 2 y los que salen del 2 van al 3.
- Cuando un servidor queda vacío selecciona al primer cliente que esté en cola. (Disciplina de cola FIFO)

En esta memoria documentaré el código fuente de la simulación, obtendré resultados analíticos para el número medio de clientes en el subsistema 3, el tiempo medio que pasa un cliente en todo el sistema y las probabilidades de que haya de 0 a 8 y que se pierda un cliente en el tercer subsistema. Finalmente, compararé las estimaciones obtenidas por la simulación y explicaré las diferencias entre los resultados de las simulaciones con tiempos de servicio exponencial y tiempos de servicio Erlang.

Documentación del programa

Procederé a explicar el código fuente del programa de cara a su comprensión y con el fin de facilitar su modificación/expansión. Haré hincapié en cuestiones técnicas y no en aquellos elementos que sirvan para dar soporte a la interfaz de usuario o métodos cuyo propósito e implementación sea trivial.

Constantes

Las constantes que se usan a lo largo de todo el programa son las siguientes:

```
#define TASALLEGADAS 2.0
#define TASASERVICIONODO1 3.0
#define TASASERVICIONODO2 4.0
#define TASASERVICIONODO3 6.0
#define LLEGADA 0
#define SALIDANODO1 1
#define SALIDANODO2 2
#define SALIDANODO3 3
#define LIBRE 0
#define OCUPADO 1
#define MAXCOLANODO3 7
#define TAMANIOMUESTRA 10000
#define PERIODOTRANSITORIO 500
#define NUMERODEMUESTRAS 25
```

Podríamos considerar las siguientes constantes como los parámetros de entrada del programa: TASALLEGADAS, TASASERVICIONODO1, TASASERVICIONODO2, TASASERVICIONODO3, MAXCOLANODO3, TAMANIOMUESTRA, PERIODOTRANSITORIO y NUMERODEMUESTRAS. En el caso de que se decidiese realizar otra simulación con distintos valores para la tasa de llegadas, tasas de servicio de cada subsistema, número máximo de clientes en la cola del subsistema 3, tamaño de la muestra, cantidad de muestras o número de clientes servidos para alcanzar el periodo transitorio, bastaría con cambiar los valores de las constantes y el programa seguiría funcionando correctamente.

A su vez y de cara a facilitar la comprensión del programa, decidí declarar las constantes LLEGADA, SALIDANODO1, SALIDANODO1, SALIDANODO1, LIBRE y OCUPADO. A medida que vayamos estudiando el código fuente se podrá apreciar que el uso de estas constantes facilita la legibilidad del código.

Variables de estado

Las siguientes variables de estado describen el estado del sistema en un periodo instante temporal:

```
//Reloj de Simulación
double reloj;
double relojTrans;
//Lista de Sucesos
double listaSucesos[4];
//Variables de Estado
int servidor1;
int servidor2;
int servidor3;
int numClientesNodo1;
int numClientesNodo2;
int colaNodo3;
double tiempoSucAnterior;
List tiempoDeLlegadasAlSistema;
```

La variable reloj es el reloj de la simulación. Lleva la cuenta del tiempo transcurrido desde el inicio de la simulación (t=0) hasta el instante actual. Ya que de cara a los resultados solo nos interesa tener en cuenta el periodo transitorio, he definido otra variable denominada relojTrans. Cuando se alcance el periodo estacionario relojTrans almacenará ese instante de tiempo, o lo que es lo mismo, cuanto tiempo ha durado el periodo transitorio (en minutos).

La lista de 4 números reales listaSucesos aloja en ella los instantes temporales en los que ocurrirán los siguientes sucesos (una llegada al subsistema 1, una salida del subsistema 1, una salida del subsistema 2 o una salida del subsistema 3). Cabe destacar que ya que nos encontramos con una red de colas en tandem, una salida un subsistema implica una llegada al contiguo. Como veremos más adelante, el reloj de la simulación obtiene su valor de la lista de sucesos.

Las variables servidor 1, 2 y 3 reflejan el estado de los servidores en un instante temporal. Tomaran valores los 0 o 1, siendo 0 si el servidor está libre y 1 si no lo está (así se definieron las constantes LIBRE y OCUPADO).

Respecto a la cantidad de clientes en los distintos nodos en un instante, las variables numClientesNodo1 y numClientesNodo2 recogen esta información. En el caso del nodo 3 nos interesa saber cuántos clientes hay en su cola ya que es finita su capacidad. La variable colaNodo3 recoge esta información.

Finalmente, la variable tiempoSucAnterior y la lista tiempoDeLlegadasAlSistema nos servirán de cara a actualizar los contadores estadísticos y posteriormente realizar estimaciones de las características de interés. La lista tiempoDeLlegadasAlSistema recoge el instante temporal en el que llega cada cliente al sistema. Cuando un cliente sale del nodo 3 o se pierde al estar llena la cola de este, se saca de la lista su tiempo de llegada sumándose dicho valor al contador estadístico tiempoClientesEnSistema y posteriormente se borra dicho elemento de la lista (con el fin de optimizar los recursos computacionales ejecutando la programación).

Contadores estadísticos

Ya que el objetivo de esta simulación es obtener las probabilidades de que haya k clientes en el subsistema 3, el número medio de clientes en el mismo y el tiempo total que pasa un cliente en todo el sistema he decidido utilizar los siguientes contadores estadísticos:

```
double tiempoClientesEnSistema;
int clientesServidosNodo3;
double tiempoServidor3ocup;
double tiempoQueHayKclientesNodo3[9];
int clientesPerdidos;
```

A medida que vayan ocurriendo los distintos sucesos posibles se irá actualizando el valor de los contadores. Cuando la muestra haya concluido, en base a los valores de los contadores se realizará la estimación de las características de interés para esa muestra y se sumarán al del resto de muestras ya realizadas. Cuando todas las muestras hayan concluido, se realizará la media de cada característica de interés en base al número de muestras (25).

Método principal (main)

```
void main() {
    double probabilidades[9],sumaL,sumaW,sumaCuadrL;
    int suceso, muestra;
    char otraVez;
    imprimeTitulo();
    pulseTeclaParaIniciar();
    do{

inicializaCaracteristicasInteres(&sumaL,&sumaW,&sumaCuadrL,probabilidades,&mu
estra);
        while (muestra <= NUMERODEMUESTRAS){
            rutinaInicializacion();
            //PERIODO TRANSITORIO
            while(clientesServidosNodo3 < PERIODOTRANSITORIO){
                suceso = rutinaTiempo();
                rutinaSucesos(suceso);
            }
            //PERIODO ESTACIONARIO
            relojTrans = reloj;
            inicializarContadoresEstadisticos();
            while (clientesServidosNodo3 < TAMANIOMUESTRA){
                suceso = rutinaTiempo();
                rutinaSucesos(suceso);
            }
            realizarEstimaciones(probabilidades, &sumaL, &sumaW,
&sumaCuadrL);
            muestra++;
        }
        generarInforme(probabilidades, sumaL, sumaW, sumaCuadrL);
        otraVez=ofrecerOtraSimulacion();
    } while (otraVez != 'N' && otraVez !='n');
}
```

Nos encontramos ante el esqueleto del programa. El programa empieza inicializando las características de interés. Esta acción solo se realiza una vez por simulación. Posteriormente empieza la simulación con muestra=1. En la rutina de inicialización se inicializan las variables de estado (1 vez por muestra). Acto seguido entramos en el periodo transitorio. Entramos en un bucle en el que primero se llama a la rutina de tiempo para actualizar el reloj y determinar el suceso que ocurre en el instante actual. A continuación, invocamos a la rutina de tiempo, pasándole como parámetro de entrada el suceso que ocurre en el instante temporal actual. Cuando el número de clientes servidos en el nodo 3 sea igual a 500 (PERIODOTRANSITORIO) saldremos de este bucle y por lo tanto habrá concluido el periodo transitorio.

Una vez concluido el periodo transitorio se guarda la duración de dicho periodo y se procede a reinicializar los contadores estadísticos a 0. Entramos en el periodo estacionario y por lo tanto volvemos a entrar en un bucle en el que se invoca a la rutina de tiempo y seguidamente a la de sucesos. Este bucle concluye cuando el número de clientes servidos en el subsistema 3 sea igual al tamaño de la muestra definido, en este caso 10.000 clientes. Una vez el programa haya salido del bucle, la simulación de la muestra habrá concluido y se procederá a realizar las estimaciones de las características de interés para esa muestra y sumarlas al cumulo de las muestras ya realizadas. Se aumenta el número de muestras realizadas y en el caso de que dicha cantidad sea menor al del número de muestras que requiere la simulación, repetiremos todo lo anterior descrito exceptuando la inicialización de las características de interés.

Cuando finalmente se realiza la cantidad de muestras determinadas por el número de muestras definido, se procede a generar y mostrar por pantalla un informe con los resultados.

Rutina de tiempo

```
int rutinaTiempo() {
    int siguiente;
    reloj = proximoEvento(&siguiente);
    return siguiente;
}
```

Se puede apreciar que la implementación de la rutina de tiempo carece de complejidad. Simplemente se basa en método auxiliar que recorre la lista de sucesos en busca del suceso cuyo instante temporal en el que ocurre sea el más próximo. El reloj se fijará con dicho instante temporal y se devolverá el tipo de suceso que ocurrirá.

La implementación de la subrutina proximoEvento() se puede ver a continuación:

```
double proximoEvento(int *siguiente) {
    double minimo = listaSucesos[0];
    *siguiente = LLEGADA;
    for (int i=1; i<4; i++)
        if (listaSucesos[i]<minimo) {
            minimo = listaSucesos[i];
            *siguiente = i;
        }
    return minimo;
}
```

Simplemente devuelve el número e índice del elemento menor de un array, aunque a efectos prácticos determina el próximo suceso y devuelve el instante temporal en el que ocurre.

Rutina de sucesos

A efectos de la simulación, la rutina de sucesos es la que se encarga de actualizar las variables de estado y los contadores estadísticos. Es el corazón de la simulación. Su implementación es la siguiente:

```
void rutinaSucesos(int suceso) {
    double exponencial;
    tiempoQueHayKclientesNodo3[servidor3+colaNodo3] += reloj-tiempoSucAnterior;
    if (servidor3 == OCUPADO)
        tiempoServidor3ocup += reloj-tiempoSucAnterior;
    switch (suceso)
    {
        case LLEGADA:
            exponencial = generadorExponencial(TASALLEGADAS);
            insertback(&tiempoDeLlegadasAlSistema,reloj);
            listaSucesos[LLEGADA] = reloj + exponencial;
            numClientesNodo1++;
            if (servidor1 == LIBRE){
                servidor1 = OCUPADO;
                exponencial = generadorExponencial(TASASERVICIONODO1);
                listaSucesos[SALIDANODO1] = reloj + exponencial;
            }
            break;
        case SALIDANODO1:
            exponencial = generadorExponencial(TASASERVICIONODO1);
            numClientesNodo1--;
            numClientesNodo2++;
            if (numClientesNodo1 != 0)
                listaSucesos[SALIDANODO1] = reloj + exponencial;
            else{
                servidor1 = LIBRE;
                listaSucesos[SALIDANODO1] = INFINITY;
            }
            if (servidor2 == LIBRE){
                servidor2 = OCUPADO;
                exponencial = generadorExponencial(TASASERVICIONODO2);
                listaSucesos[SALIDANODO2] = reloj + exponencial;
            }
            break;
        case SALIDANODO2:
            exponencial = generadorExponencial(TASASERVICIONODO2);
            numClientesNodo2--;
            if (numClientesNodo2 != 0)
                listaSucesos[SALIDANODO2] = reloj + exponencial;
            else{
                servidor2 = LIBRE;
                listaSucesos[SALIDANODO2] = INFINITY;
            }
            if (servidor3 == LIBRE){
                servidor3 = OCUPADO;
                exponencial = generadorExponencial(TASASERVICIONODO3);
                listaSucesos[SALIDANODO3] = reloj + exponencial;
            } else{
                if (colaNodo3 != MAXCOLANODO3){
                    colaNodo3++;
                } else{
                    clientesPerdidos++;
                    tiempoClientesEnSistema +=reloj -
getitem(tiempoDeLlegadasAlSistema,MAXCOLANODO3+1);
                    delete_from_position(&tiempoDeLlegadasAlSistema, MAXCOLANODO3+1);
                }
            }
            break;
        case SALIDANODO3:
            exponencial = generadorExponencial(TASASERVICIONODO3);
            clientesServidosNodo3++;
            tiempoClientesEnSistema += reloj - getitem(tiempoDeLlegadasAlSistema,0);
            delete_from_begin(&tiempoDeLlegadasAlSistema);
            if (colaNodo3 == 0) {
                servidor3 = LIBRE;
                listaSucesos[SALIDANODO3] = INFINITY;
            } else{
                colaNodo3--;
                listaSucesos[SALIDANODO3] = reloj + exponencial;
            }
            break;
    }
    tiempoSucAnterior = reloj;
```

Como se puede apreciar dependiendo del suceso que esté ocurriendo se actualizan los contadores estadísticos y variables de estado relacionadas con ese suceso. Si analizamos el código podemos llegar a la conclusión que las acciones realizadas para todos los tipos de sucesos siguen un patrón: actualizar los contadores, variables de estado y generar el siguiente suceso de su mismo tipo. Para generar el siguiente suceso podemos apreciar que se invoca a una función que genera un número aleatorio de una exponencial con parámetro correspondiente a la tasa de llegada o salida del suceso en concreto. En el caso de las salidas, siempre y cuando la cola del siguiente nodo esté vacía, también habrá que generar un tiempo de salida para el nodo al que el cliente ha transitado.

Generador de números aleatorios de una distribución exponencial

```
double generadorExponencial(double beta) {
    double numAleat = generaNumAleat();
    double exp = -1.0/beta  * log(numAleat);
    return exp;
}
```

El generador básicamente llama a una función (generaNumAleat()) que devuelve un número aleatorio de la distribución uniforme con parámetros (0,1). Luego, calcula el aleatorio de la exponencial partiendo de su parámetro y del aleatorio de la uniforme.

En el generador de números aleatorios de la U(0,1) se comprueba que el número generado sea distinto de 0.

```
double generaNumAleat() {
    double numaleat;
    do
        numaleat = rand() / (double)RAND_MAX;
    while (numaleat==0);
    return numaleat;
}
```

Estimaciones de la muestra

Como ya se ha explicado con anterioridad, la manera de realizar los cálculos de las estimaciones de las características de interés será realizando estimaciones para cada muestra y finalmente obteniendo la media de los resultados de todas las muestras.

A continuación, se muestra el método que realiza los cálculos para una muestra y los suma a los ya obtenidos en anteriores muestras:

```
void realizarEstimaciones(double probabilidades[], double *sumaL, double
*sumaW, double *sumaCuadrL) {
    double relojEstacionario = reloj-relojTrans;
    double Li=0;
    for (int i = 0; i < 9; i++){
        probabilidades[i] += tiempoQueHayKclientesNodo3[i]/relojEstacionario;
        Li += i * tiempoQueHayKclientesNodo3[i]/relojEstacionario;
    }
    *sumaL += Li;
    *sumaCuadrL += pow(Li,2);
    *sumaW += tiempoClientesEnSistema / (clientesServidosNodo3 +
clientesPerdidos);
    destroy(&tiempoDeLlegadasAlSistema); //Ya que ha concluido la muestra
destruyo la lista para no desperdiciar memoria. En la rutina de incialización
se inicializará la lista otra vez.
}
```

Generación de informe: estimaciones de la simulación

```
void generarInforme(double probabilidades[], double sumaL, double sumaW,
double sumaCuadrL) {
    double L = sumaL/NUMERODEMUESTRAS;
    double W = sumaW/NUMERODEMUESTRAS;
    double varianza = sumaCuadrL/NUMERODEMUESTRAS - pow(L,2);
    double cuasiVarianza = NUMERODEMUESTRAS/(NUMERODEMUESTRAS-1) * varianza;
    double cuasidt = sqrt(cuasiVarianza);
    double errorAbsoluto = 2.06*cuasidt/sqrt(NUMERODEMUESTRAS);
    double errorRelativo = errorAbsoluto/L;
    printf("\nSIMULACION COMPLETADA\n\n");
    imprimeAsteriscos(printf("RESUMEN PROBABILISTICO\n"));
    for (int j = 0; j < 9; ++j)
        printf("\nProbabilidad de que haya %d cliente(s) en el nodo 3 = %lf",
j, probabilidades[j]/NUMERODEMUESTRAS );
    printf("\n\nProbabilidad de que se pierda un cliente en el nodo 3 =
%lf\n",(double)clientesPerdidos/(clientesServidosNodo3+clientesPerdidos));
    imprimeAsteriscos(printf("\nPARAMETROS DE RENDIMIENTO DEL SISTEMA\n"));
    printf("\nLa estimacion de L para el nodo 3 es:\n");
    printf("L = %lf clientes\n", L);
    printf("\nLa estimacion de W para todo el sistema es:\n");
    printf("W = %lf minutos/cliente\n",W);
    imprimeAsteriscos(printf("\nINTERVALO DE CONFIANZA Y ERROR REALATIVO EN
LA ESTIMACION DE L\n"));
    printf("\nAl 95 por ciento de confianza: %lf < L < %lf",L-
errorAbsoluto,L+errorAbsoluto);
    printf("\n\nEl error relativo en la estimacion de L es de: %lf\n",
errorRelativo);
}
```

Lo primero que hace este método es calcular las estimaciones de las características de interés realizando la media de todos los obtenidos en cada muestra. Con el fin de calcular un intervalo de confianza para L y el error relativo en su estimación, realiza el cálculo de la varianza, cuasi varianza, cuasi desviación típica y el error absoluto. Muestra por pantalla los resultados obtenidos.

Modificaciones para dar soporte a una red con tiempos de servicio Erlang

Debido a que la programación se ha desarrollado de manera modularizada, las variaciones que he tenido que realizar para esta segunda simulación han sido mínimas.

Procederé a explicar los cambios realizados.

Constantes

Las únicas constantes que he tenido que modificar/añadir son las siguientes:

```
#define N_ERLANG 3
#define TASASERVICIONODO1 9.0
#define TASASERVICIONODO2 12.0
#define TASASERVICIONODO3 18.0
```

He tenido que modificar los valores de las tasas de servicio para que se ajusten a los requeridos en el enunciado del trabajo. A su vez, he añadido la constante “N_ERLANG” siendo el mismo el parámetro α de la distribución Erlang. Posteriormente se utilizará en el generador de números aleatorios de una distribución Erlang. La tasa de llegadas permanece igual que en la anterior simulación.

Rutina de sucesos

Los cambios en la rutina de sucesos con muy pequeños. En el caso de que el próximo suceso sea una llegada al subsistema 1, no he tenido que realizar ninguna modificación con respecto a la anterior simulación ya en ambos casos el tiempo entre llegadas sigue una distribución exponencial.

En los casos en los que el próximo suceso sea una salida de cualquiera de los 3 nodos, en vez de llamar al generador de la exponencial, se llama al generador de la Erlang pasándole como parámetro de entrada “N_ERLANG” y la tasa de servicio correspondiente.

En cuanto a la rutina de sucesos, no he tenido que realizar ninguna otra modificación. El código fuente es el siguiente:

```
    double erlang, exponencial;
    tiempoQueHayKclientesNodo3[servidor3+colaNodo3] += reloj-tiempoSucAnterior;
    if (servidor3 == OCUPADO)
        tiempoServidor3ocup += reloj-tiempoSucAnterior;
    switch (suceso)
    {
        case LLEGADA:
            exponencial = generadorExponencial(TASALLEGADAS);
            insertback(&tiempoDeLlegadasAlSistema,reloj);
            listaSucesos[LLEGADA] = reloj + exponencial;
            numClientesNodo1++;
            if (servidor1 == LIBRE){
                servidor1 = OCUPADO;
                erlang = generadorErlang(N_ERLANG,TASASERVICIONODO1);
                listaSucesos[SALIDANODO1] = reloj + erlang;
            }
            break;
        case SALIDANODO1:
            erlang = generadorErlang(N_ERLANG, TASASERVICIONODO1);
            numClientesNodo1--;
            numClientesNodo2++;
            if (numClientesNodo1 != 0)
                listaSucesos[SALIDANODO1] = reloj + erlang;
            else{
                servidor1 = LIBRE;
                listaSucesos[SALIDANODO1] = INFINITY;
            }
            if (servidor2 == LIBRE){
                servidor2 = OCUPADO;
                erlang = generadorErlang(N_ERLANG,TASASERVICIONODO2);
                listaSucesos[SALIDANODO2] = reloj + erlang;
            }
            break;
        case SALIDANODO2:
            erlang = generadorErlang(N_ERLANG, TASASERVICIONODO2);
            numClientesNodo2--;
            if (numClientesNodo2 != 0)
                listaSucesos[SALIDANODO2] = reloj + erlang;
            else{
                servidor2 = LIBRE;
                listaSucesos[SALIDANODO2] = INFINITY;
            }
            if (servidor3 == LIBRE){
                servidor3 = OCUPADO;
                erlang = generadorErlang(N_ERLANG, TASASERVICIONODO3);
                listaSucesos[SALIDANODO3] = reloj + erlang;
            } else{
                if (colaNodo3 != MAXCOLANODO3){
                    colaNodo3++;
                } else{
                    clientesPerdidos++;
                    tiempoClientesEnSistema +=reloj -
getitem(tiempoDeLlegadasAlSistema,MAXCOLANODO3+1);
                    delete_from_position(&tiempoDeLlegadasAlSistema, MAXCOLANODO3+1);
                }
            }
            break;
        case SALIDANODO3:
            erlang = generadorErlang(N_ERLANG, TASASERVICIONODO3);
            clientesServidosNodo3++;
            tiempoClientesEnSistema += reloj - getitem(tiempoDeLlegadasAlSistema,0);
            delete_from_begin(&tiempoDeLlegadasAlSistema);
            if (colaNodo3 == 0) {
                servidor3 = LIBRE;
                listaSucesos[SALIDANODO3] = INFINITY;
            } else{
                colaNodo3--;
                listaSucesos[SALIDANODO3] = reloj + erlang;
            }
            break;
    }
    tiempoSucAnterior = reloj;
}
```

Generador de números aleatorios de una distribución Erlang

```
double generadorErlang(int n, double beta) {
    double aleatUnif01;
    double erlang;
    double productorio = 1;
    for (int i=0; i<n; i++){
        aleatUnif01 = generaNumAleat();
        productorio *= aleatUnif01;
    }
    erlang = (-1.0)/beta  * log(productorio);
    return erlang;
}
```

La implementación se basa en aplicar las enseñanzas vistas en clase sobre como es la función para generar un número aleatorio de una distribución Erlang. No he tenido que modificar la función para generar un número aleatorio de la uniforme(0,1), simplemente se llama a dicha función tantas veces como el parámetro α de la Erlang.

Resultados

Resultados analíticos

Nos centraremos en obtener resultados analíticos para el tercer subsistema aunque calcularemos el tiempo medio que pasa un cliente en todo el sistema. Teniendo en cuenta que todas las distribuciones de tiempo entre llegadas y tiempo de servicio son exponenciales, y que la capacidad de los primeros dos nodos es infinita, podemos deducir que el tercer subsistema es un sistema M/M/1/8 con tasa de llegadas de λ =2 cliente/minuto y tiempo medio de servicio 1/6 de minuto (μ = 6 c/m).

Utilizamos el programa Qa.exe de WinQsb estableciendo los siguientes valores de entrada:

Data Description	ENTRY
Number of servers	1
Service rate (per server per minutes)	6
Customer arrival rate (per minutes)	2
Queue capacity (maximum waiting space)	7
Customer population	M
Busy server cost per minutes	
Idle server cost per minutes	
Customer waiting cost per minutes	
Customer being served cost per minutes	
Cost of customer being balked	
Unit queue capacity cost	

Los resultados son los siguientes:

01-08-2020 17:45:14 n	Estimated Probability of n Customers in the System
0	0.6667
1	0.2222
2	0.0741
3	0.0247
4	0.0082
5	0.0027
6	0.0009
7	0.0003
8	0.0001

Average number of customers in the system (L) = **0.4995**

Procederé a calcular el tiempo medio que pasa un cliente en el sistema de la siguiente manera:

$$W = \frac{L}{\sum_{i=1}^{1} \gamma_i}$$

con:

$$L = \sum_{i=1}^{3} L_i$$

y

$$\gamma: Tasa\ de\ llegadas\ desde\ el\ exterior$$

En este caso solo llegan clientes desde el exterior al subsistema 1, con una tasa de 2 clientes al minuto.

Por tanto, proseguiré calculando el número medio de clientes en el sistema (L) para los subsistemas 1 y 2:

Data Description	ENTRY
Number of servers	1
Service rate (per server per minutes)	3
Customer arrival rate (per minutes)	2
Queue capacity (maximum waiting space)	M
Customer population	M
Busy server cost per minutes	
Idle server cost per minutes	
Customer waiting cost per minutes	
Customer being served cost per minutes	
Cost of customer being balked	
Unit queue capacity cost	

Average number of customers in the system (L) = **2.0000**

Y para el subsistema 2:

Data Description	ENTRY
Number of servers	1
Service rate (per server per minutes)	4
Customer arrival rate (per minutes)	2
Queue capacity (maximum waiting space)	M
Customer population	M
Busy server cost per minutes	
Idle server cost per minutes	
Customer waiting cost per minutes	
Customer being served cost per minutes	
Cost of customer being balked	
Unit queue capacity cost	

Average number of customers in the system (L) =	1.0000

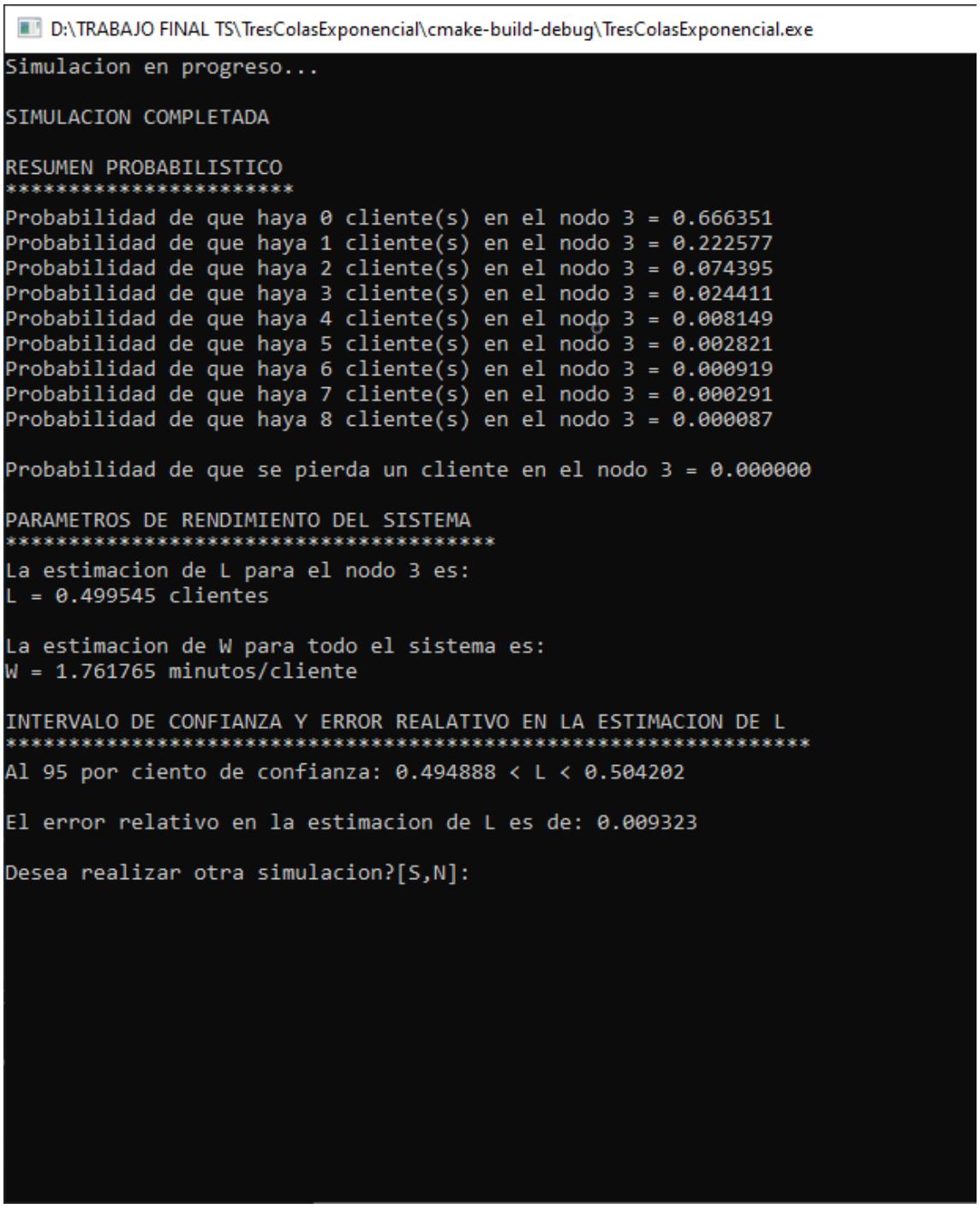
$$\therefore L = 2 + 1 + \ 0{,}4995 = 3{,}4995 \ clientes$$

Por tanto:

$$W = \frac{3{,}4995}{2} = 1{,}74975 \ minutos$$

Resultados de la simulación con tiempo de servicio exponencial

Podemos comprobar que las estimaciones obtenidas en la simulación se aproximan con bastante exactitud a los analíticos. A su vez, podemos ver que el intervalo de confianza para L que he programado contiene al resultado analítico.



Resultados de la simulación con tiempo de servicio Erlang

A continuación, se muestran las estimaciones realizadas por la simulación:

```
D:\TRABAJO FINAL TS\TresColasErlang\cmake-build-debug\TresColasErlang.exe
SIMULACION COMPLETADA

RESUMEN PROBABILISTICO
**********************
Probabilidad de que haya 0 cliente(s) en el nodo 3 = 0.666337
Probabilidad de que haya 1 cliente(s) en el nodo 3 = 0.294047
Probabilidad de que haya 2 cliente(s) en el nodo 3 = 0.036375
Probabilidad de que haya 3 cliente(s) en el nodo 3 = 0.003036
Probabilidad de que haya 4 cliente(s) en el nodo 3 = 0.000201
Probabilidad de que haya 5 cliente(s) en el nodo 3 = 0.000004
Probabilidad de que haya 6 cliente(s) en el nodo 3 = 0.000000
Probabilidad de que haya 7 cliente(s) en el nodo 3 = 0.000000
Probabilidad de que haya 8 cliente(s) en el nodo 3 = 0.000000

Probabilidad de que se pierda un cliente en el nodo 3 = 0.000000

PARAMETROS DE RENDIMIENTO DEL SISTEMA
**************************************
La estimacion de L para el nodo 3 es:
L = 0.376729 clientes

La estimacion de W para todo el sistema es:
W = 1.299317 minutos/cliente

INTERVALO DE CONFIANZA Y ERROR REALATIVO EN LA ESTIMACION DE L
**************************************************************
Al 95 por ciento de confianza: 0.374632 < L < 0.378827

El error relativo en la estimacion de L es de: 0.005567

Desea realizar otra simulacion?[S,N]:
```

En este caso, y como ocurre frecuentemente en las simulaciones, nos es relativamente complicado saber si los resultados obtenidos son correctos ya que no podemos obtener resultados analíticos. Sin embargo, procederé a compararlos con los de la exponencial y a intentar explicar a qué se pueden deber las diferencias.

Podemos apreciar que el número medio de clientes en el tercer nodo (L) es menor que en el caso de la exponencial. Esto concuerda con las probabilidades obtenidas ya que en toda la simulación nunca ha llegado a haber más de 5 clientes en la cola. A su vez, el tiempo medio que pasan los clientes en el sistema también es menor que en el caso de la exponencial. Todo lo anterior indica que los clientes son servidos más rápido en cada nodo que en la simulación con tiempos de servicio exponencial. Es más, todas las tasas de servicio de los nodos son superiores a los de la simulación anterior (9, 12 y 18 frente a 3,4 y 6, suma de 3 exponenciales).

Analizando los generadores de números aleatorios de ambas distribuciones, sabiendo que la distribución Erlang es una suma de exponenciales (en este caso 3). Sin embargo, ya que debemos obtener el producto de 3 números de la uniforme(0,1), el número que generemos debería ser menor que en el caso de la simulación con tiempos de servicio exponenciales.

Por tanto, ya que las tasas de servicio son erlang de 3 veces las distribuciones exponenciales con tasas de 3,4 y 6, podemos concluir que los clientes son servidos significativamente más rápido.