Universidade Federal Fluminense
Instituto de Computação
Ciência da Computação

Projeto de AD

Simulador de filas

Lucas Pontes Siqueira
Rafael Duarte Campbell De Medeiros
Dezembro de 2018

Sumário

1 Introdução

A ideia inicial era simular algo como as filas de caixa rápida que algumas redes de mercado adotaram, onde existe uma única fila que dá acesso a uma bateria de caixas rápidos. Como foi indicado pelo professor, num primeiro momento se buscou modelar uma fila com apenas um servidor, posteriormente se fez um sistema que permitia tratar com vários servidores. Como proposto na segunda parte do projeto, foi desenvolvido na linguagem java um simulador onde os elementos chegam em uma fila e se dirigem a uma quantidade previamente escolhida de caixas. Para cada simulação, o usuário deve indicar parâmetros como:

- tamanho da fila
- tempo médio entre as chegadas
- tempo médio de serviço
- número de servidores
- política da fila (FIFO ou LIFO)
- tempo de simulação

Durante a chamada ao método, deve-se colocar os parâmetros na ordem pré-estabelecida. O simulador também gera um gráfico de tempo médio de espera em função da taxa de chegada (1/E[C]) para os valores propostos no experimento 3. Vale ressaltar que, para o simulador, adotou-se o minuto como tempo padrão, então todos os valores de tempo são dados em minutos.

2 Gerador de números pseudo-aleatórios

Para gerar os valores aleatórios, foi usado o método congruente linear, adotando os valores $M = 255$, $a = 359753158$ e $c = 35$. Para semente, usou-se o tempo de relógio do sistema em milissegundos.

$$x_i = ((a * x_{i-1}) + c)MODm$$

2.1 Código para classe GeradorAleatorio

```
1  public class GeradorAleatorio {
2      private final double m = 255;
3      private double a = 359753158;
4      private double c = 35;
5      private double ultimoValor;
6
7      public GeradorAleatorio(){
8          this.ultimoValor =  System.currentTimeMillis(); //pega o horario do sistema
9      }
10
11     public double geraValor(){
12         double val =  (((this.a * this.ultimoValor) + this.c) % this.m);
13         this.ultimoValor = val;
14         return val;
15     }
16 }
```

3 Gerador de valores exponenciais

Para garantir que a chegada e atendimento na fila seguissem os resultados de uma variável aleatória com distribuição exponencial, usou-se a fórmula proposta em sala, onde λ é a taxa e u um valor aleatório.

$$x = -\frac{1}{\lambda} log(u)$$

Não foi implementada uma classe exclusiva para os geradores, apenas se usa a fórmula para calcular o tempo de chegada e de serviço dentro da classe do elemento.

4 Elemento

O elemento é, basicamente, o objeto que chega à fila. Ele tem como atributos um tempo de chegada, tempo de serviço e um tempo de saída. Para isso, assim que um objeto chega a fila, cria-se uma nova instância de elemento que recebe o tempo atual como parâmetro; o elemento, então, gera um valor exponencial para sua chegada e a soma com o tempo atual, criando o tempo de sua chegada. Quando é atendido por um servidor, gera um valor exponencial para seu tempo de serviço que, somado ao tempo atual, determina o momento em que sairá do sistema.

4.1 Código da classe Elemento

```
1  public final class Elemento {
2
3      private double tempoChegada;
4      private double tempoServico;
5      private double tempoSaida;
6
7      public Elemento(GeradorAleatorio gerador, double taxaServico,
8              double taxaChegada, double tempo){
9          this.tempoChegada = geraChegada(gerador, taxaChegada) + tempo;
10         this.tempoServico = geraServico(gerador, taxaServico);
11     }
12
13     public double geraServico(GeradorAleatorio gerador, double taxaServico){
14         double exp = (Math.log(gerador.geraValor())/-taxaServico);
15         return(Math.abs(exp));
16     }
17
18     public double geraChegada(GeradorAleatorio gerador, double taxaChegada){
19         double exp = (Math.log(gerador.geraValor())/-taxaChegada);
20         return(Math.abs(exp));
21     }
22 }
```

5 Lista de eventos (*backlog*)

Como foi pedido, foi implementada uma espécie de *backlog* de eventos, que guarda um nome (como "entrada"ou "saída") e um tempo. Ao longo da execução, cada chegada adiciona um evento de entrada para o histórico e, para cada saída, cria-se um evento de saída. Pode-se chamar uma função que imprime o *backlog* na tela, indicando em ordem as entradas e saídas e os respectivos tempos.

5.1 Código da classe Evento

```
public class Evento {

    private final String tipo;
    private final double tempo;

    public Evento(String tipo, double tempo){
        this.tipo = tipo;
        this.tempo = tempo;
    }
}
```

6 Simulação

Para a simulação, foi necessário criar uma classe que recebe muitos parâmetros como atributos e um método para construir a simulação.

6.1 Atributos da classe SimuladorDeFilas

```
public class SimuladorDeFilas {

    private double esperancaW;
    private double esperancaN;
    private double little;
    private double descarteTotal;
    private double utilizacao;
    private final double utilizacaoFormula;
    private final int tamanhoLimiteFila;
    private final double taxaChegada;
    private final double taxaServico;
    private final double tempoServico;
    private final int numServidores;
    private final int tipoAtendimento;
    private final double tempoLimite;

    private ArrayList<Evento> listaEventos;

    private final int numeroExperimento;
}
```

6.2 Método de simulação

O método principal, que constroi a simulação, acabou ficando bastante extenso. O que ele faz é operar o tempo da execução como se cada 0.0001 minuto fosse um momento, representado por um loop. Ao percorrer o tempo, vai-se observando os tempos dos eventos de entrada e saída e o tempo atual, para que no momento correto os elementos sejam colocados ou excluídos da lista.

```java
    public void simula() {

        double minutos = 0.0;
        double fimAtendimento = 0.0;

        //para as medidas de interesse
        int elementos = 0;
        double tempoTotalServico = 0;
        int totalSaidas = 0;

        //gerador aleatorio
        GeradorAleatorio gerador = new GeradorAleatorio();

        //tempo medio de espera
        double tempoEsperaTotal = 0;
        int qtdTempoEspera = 0;


        //fila
        ArrayList<Elemento> filaElementos = new ArrayList<>();
        ArrayList<Elemento> listaAtendendo = new ArrayList<>();
        ArrayList<Elemento> remover = new ArrayList<>();
        int tamanhoLista = 0;

        //lista de eventos
        listaEventos = new ArrayList<>();

        //gera os primeiro caso de entrada e saida
        Elemento elem = new Elemento(gerador, taxaServico, taxaChegada, minutos);

        //numero medio de elementos
        int totalElementos = 0;
        double tempoAtual = 0;

        //utilizacao
        double tempoUtilizado = 0;


        while(minutos < tempoLimite){
            if(elem.getTempoChegada() <= minutos){ //processando chegadas na fila
                if(filaElementos.size() < tamanhoLimiteFila){
                    filaElementos.add(elem);
                    elementos += 1;
                    listaEventos.add(new Evento("chegada", minutos));
                }else{ //conta os descarte
                    descarteTotal += 1;
                }
                elem = new Elemento(gerador, taxaServico, taxaChegada, minutos);
            }

            if(listaAtendendo.size() < numServidores){ //verifica se tem servidor
                livre
                if(!filaElementos.isEmpty()){
                    int pos;
                    if(tipoAtendimento == 0){ //seguindo politica FIFO
                       pos = 0;
                    }else{ //seguindo politica LIFO
                        pos = filaElementos.size()-1;
                    }
```

```
                    filaElementos.get(pos).setTempoSaida(minutos);
                    listaAtendendo.add(filaElementos.get(pos));
                    filaElementos.remove(pos);
                }
            }
            for(Elemento ele : listaAtendendo){ //processa a saida dos atendimentos
                if(ele.getTempoSaida() <= minutos){
                    elementos -= 1;
                    remover.add(ele);
                    tempoEsperaTotal += ele.getTempoSaida() - ele.getTempoChegada()
                        ;
                    qtdTempoEspera += 1;
                    listaEventos.add(new Evento("saida", minutos));
                }
            }
            remover.forEach((ele) -> { //remove os elementos que ja sairam da lista
                listaAtendendo.remove(ele);
            });
            remover.clear();

            if(tempoAtual >= 1.0){
                totalElementos += elementos;
                tempoAtual = 0;
            }
            tempoUtilizado += (listaAtendendo.size()) * 0.001;

            tempoAtual += 0.001;
            minutos += 0.001; //atualiza o tempo

        }
        esperancaW = tempoEsperaTotal/qtdTempoEspera;
        esperancaN = totalElementos/tempoLimite;
        utilizacao = tempoUtilizado/(tempoLimite*numServidores);
    }
```

6.3 Medidas de interesse da simulação

Esta mesma classe também permite imprimir em tela algumas medidas de interesse, como:

- tempo médio de espera
- numero médio de pessoas no sistema
- utilização conseguida no sistema
- utilização pela fórmula ($\lambda * E[X]$)
- taxa de descarte

```
    public void imprimeMedidas(){
        System.out.println("\n\nExperimento "+Integer.toString(numeroExperimento));
        System.out.println("Esperanca de W: "+Double.toString(esperancaW));
        System.out.println("Esperanca de N: "+Double.toString(esperancaN));
        System.out.println("Taxa de descarte: "+Double.toString(descarteTotal/60));
        System.out.println("Utilizacao: "+Double.toString(utilizacao));
        System.out.println("Utilizacao por formula: "+Double.toString(
            utilizacaoFormula));
    }
```

6.4 *backlog* dos eventos

Também há o método que permite imprimir em tela o *backlog* dos eventos, seguindo um padrão

tipo -> tempo

```
    public void imprimeEventos(){
        for(Evento ev : listaEventos){
            System.out.println(ev.getTipo()+" -> "+Double.toString(ev.getTempo()));
        }
    }
```

7 Classe Main

A classe Main é, de modo superficial, a interação com o usuário. Ela é responsável por recolher as informações da simulação que se pretende fazer, devolver as medidas de interesse e, eventualmente, imprimir os eventos.

7.1 Código da classe Main

```
public class Main {
    public static void main(String[] args){//recebendo os parametros da fila
        Scanner in = new Scanner(System.in);
        System.out.print("Tamanho da fila: ");
        final int tamanhoLimiteFila = in.nextInt();
        System.out.print("Tempo medio de chegada: ");
        final double tempoChegada = in.nextDouble();
        System.out.print("Tempo medio de servico: ");
        final double tempoServico = in.nextDouble();
        System.out.print("Quantos servidores: ");
        final int numServidores = in.nextInt();
        System.out.print("FIFO(0) ou LIFO(1): ");
        final int tipoAtendimento = in.nextInt();
        System.out.print("Tempo de simulacao: ");
        final int tempoLimite = in.nextInt();
        SimuladorDeFilas simulador = new SimuladorDeFilas(tamanhoLimiteFila,
            tempoChegada, tempoServico, numServidores, tipoAtendimento, tempoLimite
            , 1);
        simulador.imprimeMedidas();
        System.out.println("\n\nDeseja imprimir backlog de eventos: (s/n)");
        if(in.next().toLowerCase().equals("s")){
            System.out.println("\n");
            simulador.imprimeEventos();
        }
    }
}
```

7.2 Exemplo de execução da classe Main

Um exemplo de execução tem, por exemplo, as seguintes entradas.

```
Tamanho da fila: 15
Tempo médio de chegada: 0,0025
Tempo médio de servico: 0,0032
Quantos servidores: 2
FIFO(0) ou LIFO(1): 0
Tempo de simulação: 60

Experimento 1
Esperanca de W: 0.015028746115486071
Esperanca de N: 1.3333333333333333
Taxa de descarte: 0.0
Utilização: 0.6098583333332993
Utilização por fórmula: 0.64

Deseja imprimir backlog de eventos: (s/n)
n
```

8 Geração dos gráficos

Para geração de gráficos, foi utilizada a biblioteca para java *JFreeChart*, que permite criar gráficos das mais variadas formas bastante intuitivamente. Para o gráfico, foi utilizada apenas uma estrutura para armazenar os gráficos e uma classe que cria o gráfico e o armazena como arquivo .png na pasta do projeto.

8.1 Classe para geração do gráfico

```
1  public class GeraGrafico {
2
3      public static void GeraGrafico(DefaultCategoryDataset ds, String x, String y,
           String nomeArquivo){
4
5          // cria o grafico
6          JFreeChart grafico = ChartFactory.createLineChart("", x,
7              y, ds, PlotOrientation.VERTICAL, false, true, false);
8
9          OutputStream arquivo = null;
10         try {
11             arquivo = new FileOutputStream(nomeArquivo+".png");
12         } catch (FileNotFoundException ex) {
13             Logger.getLogger(GeraGrafico.class.getName()).log(Level.SEVERE, null,
                   ex);
14         }
15         try {
16             ChartUtilities.writeChartAsPNG(arquivo, grafico, 550, 400);
17         } catch (IOException ex) {
18             Logger.getLogger(GeraGrafico.class.getName()).log(Level.SEVERE, null,
                   ex);
19         }
20     }
21 }
```

9 Execução do experimento 3

Para o experimento, foi pedido que se fizesse a simulação para vários valores de tempos entre chegadas e se criasse um gráfico do tempo médio de espera em função da taxa de chegada, que é dada por 1/E[C]. Para isso, foi criada também uma classe que contém apenas um método estático para execução desse experimento.

9.1 Código para execução do experimento

```
1  public class Experimento3 {
2
3      public static void executa(){
4          Data data = new Data();
5          //200 milissegundos -> 0.00333 minutos
6          SimuladorDeFilas simulador = new SimuladorDeFilas(15, 0.00333, 0.0015, 1,
               0, 60, 1);
7          simulador.imprimeMedidas();
8          data.adiciona(simulador.getEsperancaW(), "", Double.toString(simulador.
               getTaxaChegada()));
9          //180 milissegundos -> 0.003 minutos
10         simulador = new SimuladorDeFilas(15, 0.003, 0.0015, 1, 0, 60, 2);
11         simulador.imprimeMedidas();
12         data.adiciona(simulador.getEsperancaW(), "", Double.toString(simulador.
               getTaxaChegada()));
13         //160 milissegundos -> 0.002666 minutos
14         simulador = new SimuladorDeFilas(15, 0.002666, 0.0015, 1, 0, 60, 3);
15         simulador.imprimeMedidas();
16         data.adiciona(simulador.getEsperancaW(), "", Double.toString(simulador.
               getTaxaChegada()));
17         //140 milissegundos -> 0.002333 minutos
18         simulador = new SimuladorDeFilas(15, 0.002333, 0.0015, 1, 0, 60, 4);
19         simulador.imprimeMedidas();
20         data.adiciona(simulador.getEsperancaW(), "", Double.toString(simulador.
               getTaxaChegada()));
21         //120 milissegundos -> 0.002 minutos
22         simulador = new SimuladorDeFilas(15, 0.002, 0.0015, 1, 0, 60, 5);
23         simulador.imprimeMedidas();
24         data.adiciona(simulador.getEsperancaW(), "", Double.toString(simulador.
               getTaxaChegada()));
25         //100 milissegundos -> 0.001666 minutos
26         simulador = new SimuladorDeFilas(15, 0.001666, 0.0015, 1, 0, 60, 6);
27         simulador.imprimeMedidas();
28         data.adiciona(simulador.getEsperancaW(), "", Double.toString(simulador.
               getTaxaChegada()));
29
30         GeraGrafico.GeraGrafico(data, "E[W]", "Taxa de chegada", "graficoExercicio"
               );
31     }
32 }
```

9.2 Exemplo de resposta no terminal

```
Experimento 1
Esperanca de W: 0.00813899906738676
Esperanca de N: 16.0
Taxa de descarte: 64.91666666666667
Utilização: 0.9996333333325153
Utilização por fórmula: 0.4504504504504505

Experimento 2
Esperanca de W: 0.007202022171520134
Esperanca de N: 0.6666666666666666
Taxa de descarte: 0.0
Utilização: 0.523700000000257
Utilização por fórmula: 0.5

Experimento 3
Esperanca de W: 0.007374260252114393
Esperanca de N: 1.0
Taxa de descarte: 0.0
Utilização: 0.5932166666667958
Utilização por fórmula: 0.5626406601650412

Experimento 4
Esperanca de W: 0.008160761977311038
Esperanca de N: 0.7333333333333333
Taxa de descarte: 0.0
Utilização: 0.6589999999999758
Utilização por fórmula: 0.6429489927132448

Experimento 5
Esperanca de W: 0.0071933636100827095
Esperanca de N: 0.7333333333333333
Taxa de descarte: 0.0
Utilização: 0.7393999999997884
Utilização por fórmula: 0.75

Experimento 6
Esperanca de W: 0.121845443769653
Esperanca de N: 14.9
Taxa de descarte: 3.2333333333333334
Utilização: 0.8795666666661285
Utilização por fórmula: 0.9003601440576231
```

9.3 Gráfico gerado em .png