Universidade Federal do Rio de Janeiro
Departamento de Ciência da Computação

Trabalho de Simulação

Autores:
Marco Vinícius Lima Reina de Barros
Pedro Henrique Pereira de Jesus
Ronald Andreu Kaiser

Todos os membros do grupo participaram da implementação do código e da documentação do relatório.

28 de Novembro de 2010

Conteúdo

Capítulo 1

Introdução

1.1 Funcionamento geral

O simulador possui uma lista de eventos que é processada continuamente, até alcançar um número máximo de clientes que desejamos atender por rodada.

São executadas tantas rodadas quanto forem necessárias até todos os intervalos de confiança dos valores que estão sendo estimados forem válidos, ou seja, <= 10% da média do estimador.

Inicialmente, calculamos o tempo de chegada do primeiro cliente que representa um evento de chegada no sistema. A passsagem de um cliente pelo sistema possibilita a criação dos seguintes eventos:

- <tempo, tipo: chegada no sistema>
- <tempo, tipo: entrada no servidor pela primeira vez>
- <tempo, tipo: saída do servidor>
- <tempo, tipo: entrada no servidor pela segunda vez>

Quando um evento de chegada ocorre, outro evento de chegada é criado com o tempo definido com o tempo de chegada baseado em uma distribuição exponencial, que representa o tempo de chegada do próximo cliente. Deste modo, os clientes vão chegando no sistema e a lista de eventos é processada.

Quando um evento é processado, ele é removido da lista de eventos e os novos eventos gerados a partir deste são criados e adicionados na lista, ordenada pelos tempos em que cada evento ocorre.

Todos os parâmetros, descritos na seção 1.7 são passados para o simulador em sua inicialização.

1.2 Estruturas internas utilizadas

Para viabilizar a implementação da ideia geral apresentada acima, dividimos o simulador em alguns módulos, abaixo estão explicitados os mais importantes:

Módulos utilitários:

- Estimator: Módulo que possui métodos para retornar os estimadores de média, variância e calcula intervalos de confiança.
- Dist: Módulo com o método que retorna os tempos aleatórios de chegada de uma distribuição exponencial.

Classes:

- Client: classe que representa um cliente que entra no sistema. Possui seus tempos de entrada e saída da fila, tempo no servidor e cor.
- EventHeap: classe que representa a lista de eventos que é processada durante uma rodada de simulação.
- Simulator: classe que implementa a lógica principal do simulador, processa as rodadas tratando os eventos e as chegadas dos clientes. E calcula as estimativas das variáveis aleatórias.
- Analytic: classe que serve para calcular os resultados de forma analítica.

1.3 Linguagem de Programação

Para a codificação do simulador foi utilizada a linguagem de programação Python, versão 2.5.5.

1.4 Geração de variáveis aleatórias

A linguagem Python utiliza o gerador de números aleatórios "Mersenne Twister", um dos métodos mais extensivamente testados existentes.

O método garante que a sequência de números gerados pela chamada random() só se repetirá em um período de $2^{19937} - 1$. Como o período é bem extenso, não precisamos nos preocupar com redefinir seeds que gerassem sequências sobrepostas.

A semente inicial utilizada pelo gerador, por default, é o timestamp corrente no momento do import do módulo random.

1.5 Métodos utilizados

Foi utilizado o método replicativo para a simulação.

1.6 Implementação do conceito de cores

O conceito de cores foi implementando adicionando o atributo "color"no objeto Client, que possui 2 valores: TRANSIENT ou EQUILIBRIUM. O número de clientes que representam a fase transiente são associados à cor TRANSIENT e os outros clientes são associados à cor EQUILIBRIUM.

Ao final da rodada de simulação os clientes que possuem a cor TRANSIENT são descartados do cálculo dos estimadores.

1.7 Escolha dos parâmetros

Ao iniciar o simulador, são executados em sequência todas as simulações necessárias para obtermos todos os dados requeridos para ambas as políticas de atendimento com os parâmetros:

- $\rho = 0.2$ - # de clientes na frase transiente = 30000
- $\rho = 0.4$ - # de clientes na frase transiente = 40000
- $\rho = 0.6$ - # de clientes na frase transiente = 80000
- $\rho = 0.8$ - # de clientes na frase transiente = 400000
- $\rho = 0.9$ - # de clientes na frase transiente = 500000

A escolha do número de clientes da fase transiente para cada utilização foi estimada de acordo com o que é exposto no capítulo 3.

O número de clientes que são avaliados a cada rodada, ou seja, pertencentes à fase de equilíbrio do sistema, é um parâmetro de entrada para o simulador. Para o cálculo dos resultados foram utilizados apenas os dados de 100.000 clientes, sem contar os presentes na fase transiente.

1.8 Máquina utilizada

As configurações da máquina utilizada para executar a simulação e os tempos de cada experimento são mostrados abaixo:

Configurações:

- Processador: Intel Core Duo 2 GHz
- Memória: 2GB DDR 2 667Mhz
- Sistema Operacional: MAC OS X 10.5.8 (Leopard)

Duração dos experimentos:

- $\rho = 0.2$ - F.C.F.S : 24.88s.
- $\rho = 0.2$ - L.C.F.S : 19.15s.
- $\rho = 0.4$ - F.C.F.S : 68.93s.
- $\rho = 0.4$ - L.C.F.S : 27.58s.
- $\rho = 0.6$ - F.C.F.S : 120.42s.
- $\rho = 0.6$ - L.C.F.S : 27.97s.
- $\rho = 0.8$ - F.C.F.S : 627.45s.
- $\rho = 0.8$ - L.C.F.S : 1037.33s.
- $\rho = 0.9$ - F.C.F.S : 3403.95s.
- $\rho = 0.9$ - L.C.F.S : 4732.28s.

Capítulo 2

Teste de Correção

Nesta seção você descreverá os testes de correção que foram efetuados para garantir o pleno funcionamento do simulador. Você deve demonstrar que o seu programa está simulando exatamente e com correção o esquema proposto. As fórmulas analíticas não podem ser utilizadas para garantir a correção. Servem apenas de orientação, pois na maioria das vezes partimos para a simulação exatamente por não termos os resultados analíticos. Procure rodar o simulador com cenários determinísticos com estatística conhecida, demonstrando que o programa está correto. Você deverá anexar comentários sobre a boa qualidade dos intervalos de confiança obtidos e como os valores exatos se encaixam nestes intervalos, para os diversos valores de r.

Capítulo 3

Estimativa da fase transiente

Os valores usados para a fase transiente de cada experimento foram estimados desenhando gráficos mostrando a geração dos valores da variância do tempo de espera na fila 2 (V(W2)) em 5 rodadas.

Esse estimador foi usado pois ele é o que converge mais lentamente para o intervalo de confiança válido. Este fato foi comprovado executando o simulador um número considerável de vezes para cada tipo de utilização do servidor.

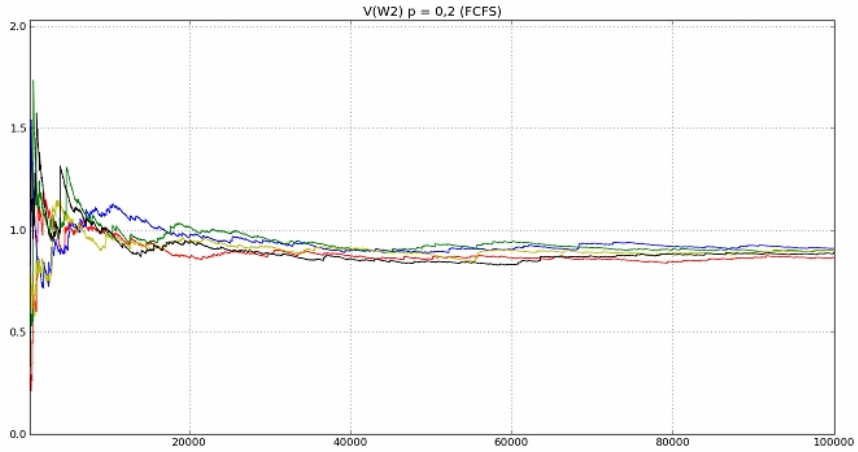
Foram gerados diversos gráficos para cada utilização, e em cada um deles a semente é diferente, já que ela é definida como está exposto na seção 1.4. Todos mostraram o mesmo comportamento. Escolhemos um de cada tipo de experimento para mostrar no relatório.

Com os gráficos gerados foi possível ter uma boa noção de que em que ponto o simulador começa a entrar na fase de equilíbrio. Eles são mostrados na seção 3.2.

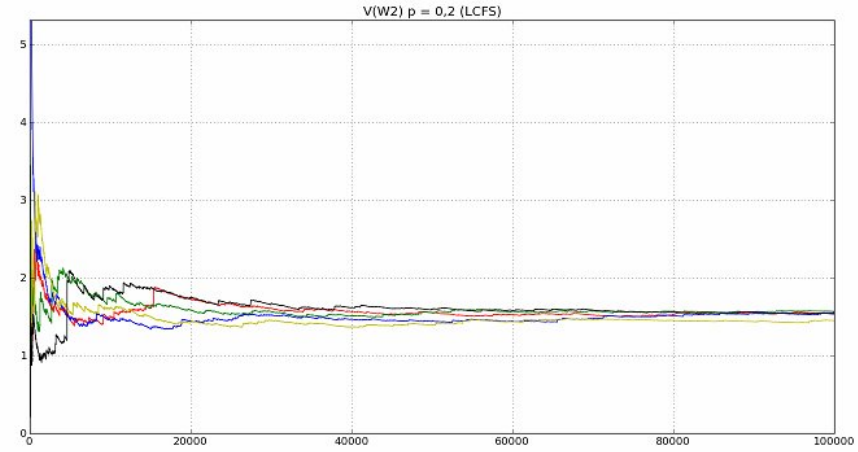
3.1 No simulador:

Dentro do simulador esses valores de fase transiente para cada experimento são dados em uma lista junto com os valores das taxas de entrada.

3.2 Gráficos:

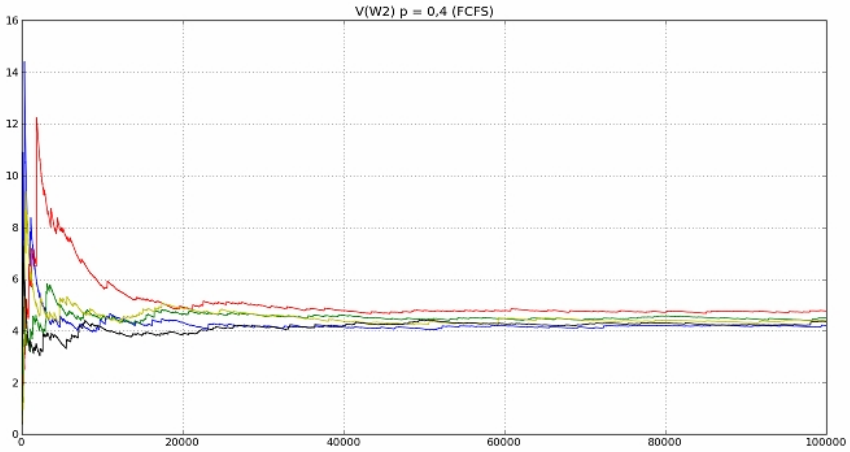


(a) First Come First Served

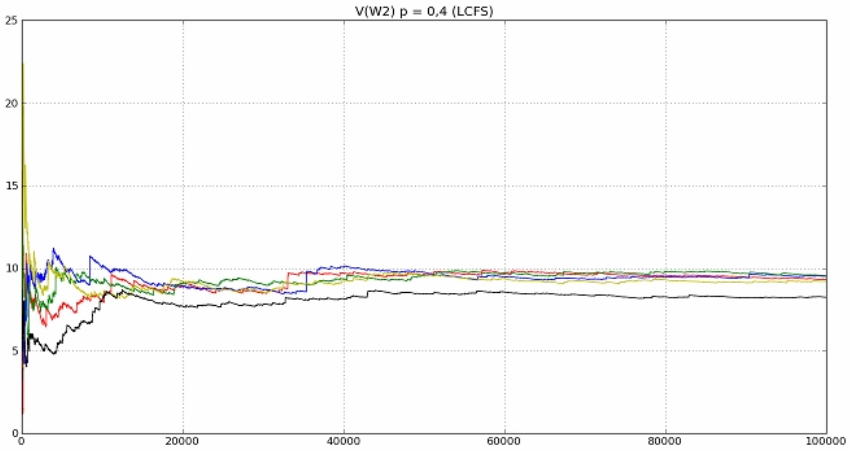


(b) Last Come First Served

Figura 3.1: V(W2) para $\rho = 0.2$. Nesse caso identificamos que a fase de equilíbrio começa quando aproximadamente 30.000 clientes já passaram pelo sistema.

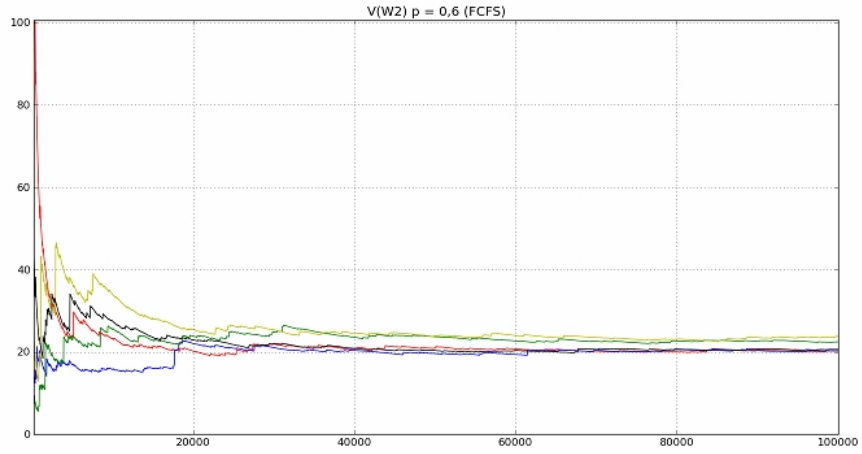


(a) First Come First Served

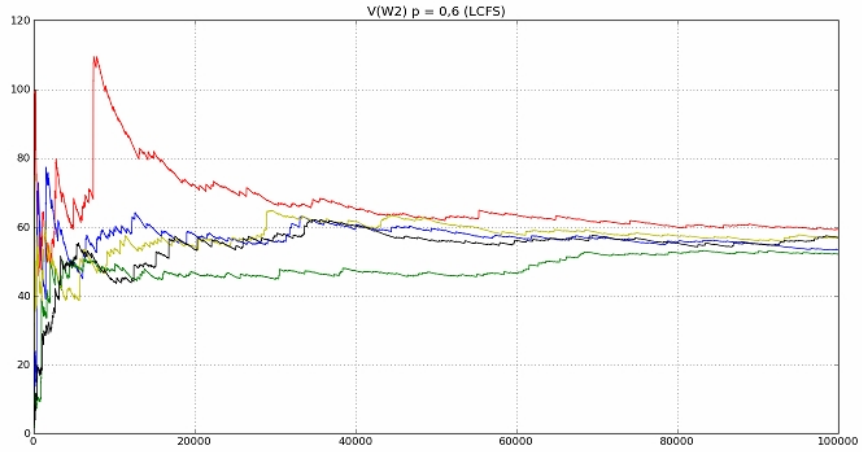


(b) Last Come First Served

Figura 3.2: V(W2) para $\rho = 0.4$. Nesse caso identificamos que a fase de equilíbrio começa quando aproximadamente 40.000 clientes já passaram pelo sistema.

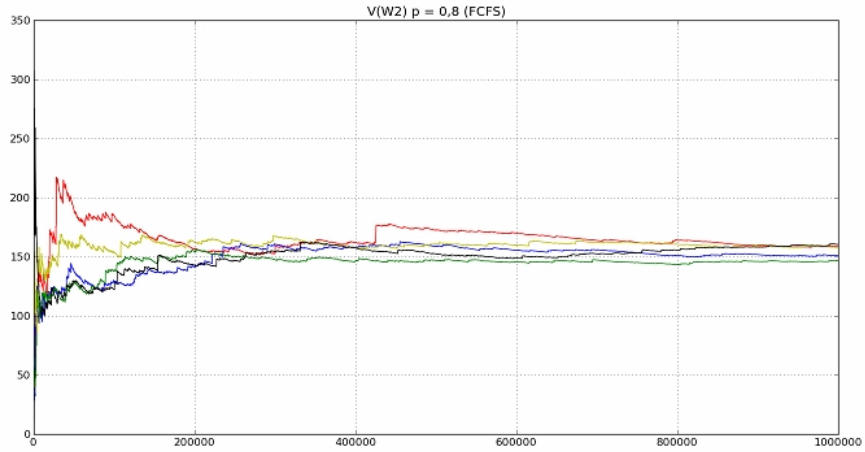


(a) First Come First Served

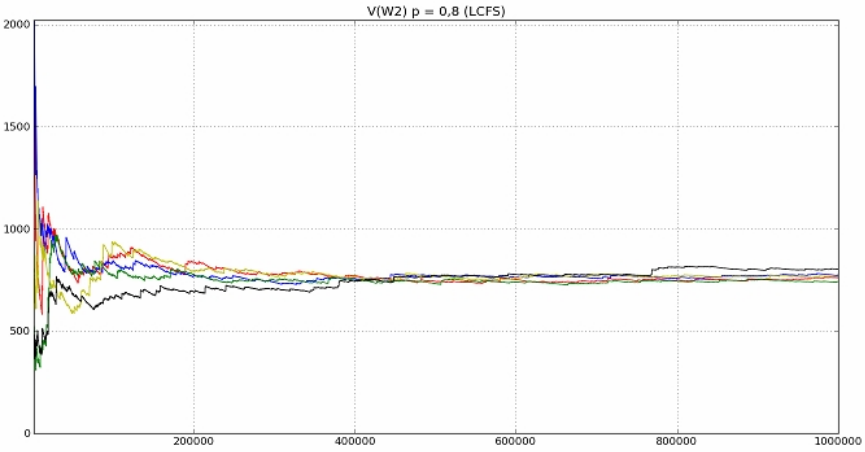


(b) Last Come First Served

Figura 3.3: V(W2) para $\rho = 0.6$. Nesse caso identificamos que a fase de equilíbrio começa quando aproximadamente 80.000 clientes já passaram pelo sistema.

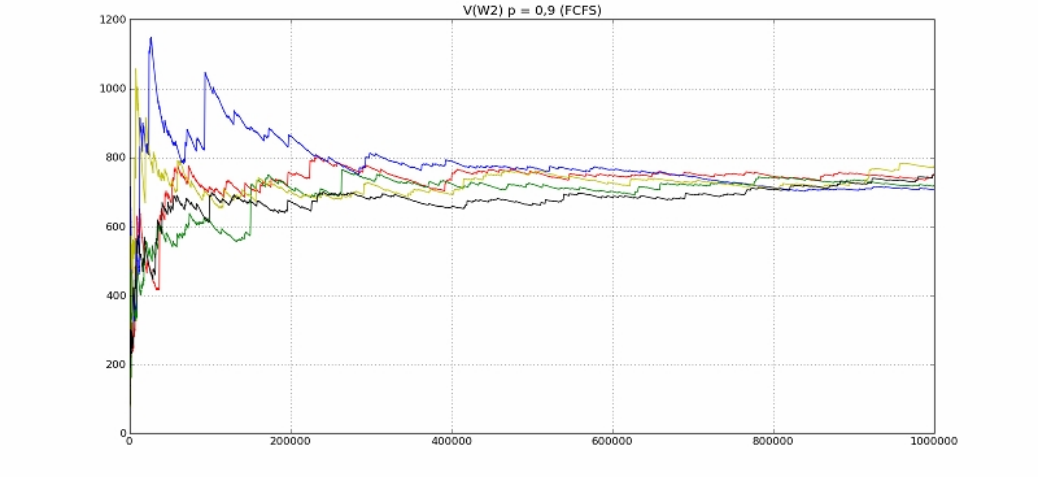


(a) First Come First Served

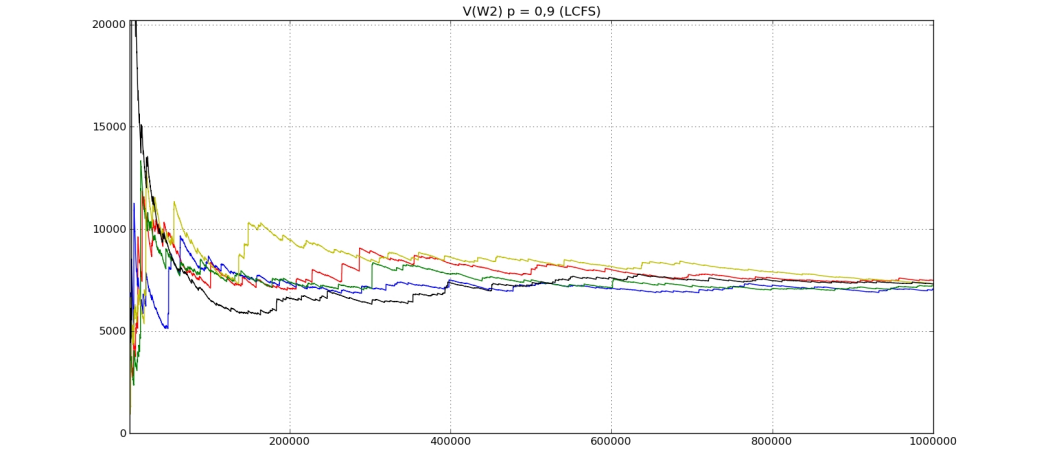


(b) Last Come First Served

Figura 3.4: V(W2) para $\rho = 0.8$. Nesse caso identificamos que a fase de equilíbrio começa quando aproximadamente 400.000 clientes já passaram pelo sistema.



(a) First Come First Served



(b) Last Come First Served

Figura 3.5: V(W2) para $\rho = 0.9$. Nesse caso identificamos que a fase de equilíbrio começa quando aproximadamente 500.000 clientes já passaram pelo sistema.

Capítulo 4

Resultados

Neste capítulo apresentamos os resultados obtidos na seção 4.1 e os comentários a respeito deles na seção 4.2.

4.1 Tabelas

Os resultados gerados pelo simulador são mostrados nas duas tabelas abaixo. O formato delas segue o seguinte padrão:

- Cada coluna representa um tipo de resultado (Tempo de espera na fila 1, etc.).
- Cada linha representa uma forma de utilização do servidor diferente.
- Cada célula contém respectivamente o valor analítico do resultado, o valor estimado pelo simulador e o tamanho do intervalo de confiança em % do valor estimado.

Tabela com os resultados para a politica de atendimento F.C.F.S										
uti.	**E[N1]**	**E[N2]**	**E[T1]**	**E[T2]**	**E[Nq1]**	**E[Nq2]**	**E[W1]**	**E[W2]**	**V(W1)**	**V(W2)**
0.2	0.12222	0.13056	1.22222	1.30556	0.02222	0.03056	0.22222	0.30556	0.44444	X
	0.12235	0.13034	1.22245	1.30357	0.02232	0.03048	0.22284	0.30408	0.45328	0.89774
	1.94%	2.64%	1.55%	2.31%	3.5%	5.98%	4.11%	6.5%	6.73%	9.38%
0.4	0.3	0.4	1.5	2.0	0.1	0.2	0.5	1.0	1.0	X
	0.29969	0.39933	1.49965	2.00103	0.09962	0.19957	0.49885	1.00104	0.98963	4.52897
	0.73%	1.63%	0.58%	1.87%	1.55%	2.9%	1.52%	3.46%	2.85%	8.97%
0.6	0.55714	1.13571	1.85714	3.78571	0.25714	0.83571	0.85714	2.78571	1.71429	X
	0.55845	1.13964	1.86446	3.81291	0.25813	0.83965	0.86328	2.81424	1.73843	22.21279
	0.71%	1.6%	0.7%	2.51%	1.08%	2.12%	1.21%	3.39%	2.4%	9.74%
0.8	0.93333	4.13333	2.33333	10.33333	0.53333	3.73333	1.33333	9.33333	2.66667	X
	0.93344	4.12212	2.33201	10.29418	0.53334	3.72216	1.33238	9.29394	2.66799	150.76599
	0.38%	1.34%	0.53%	3.0%	0.54%	1.48%	0.9%	3.32%	2.13%	9.85%
0.9	1.18636	11.12727	2.63636	24.72727	0.73636	10.67727	1.63636	23.72727	3.27273	X
	1.18532	11.06688	2.63626	24.62149	0.73556	10.61707	1.63607	23.62118	3.27872	766.03746
	0.17%	1.24%	0.39%	3.0%	0.24%	1.29%	0.59%	3.12%	1.52%	9.99%

Figura 4.1: Tabela com os valores para a política de atendimento First Come First Served.

Tabela com os resultados para a politica de atendimento L.C.F.S										
uti.	**E[N1]**	**E[N2]**	**E[T1]**	**E[T2]**	**E[Nq1]**	**E[Nq2]**	**E[W1]**	**E[W2]**	**V(W1)**	**V(W2)**
0.2	0.12222	0.13056	1.22222	1.30556	0.02222	0.03056	0.22222	0.30556	0.49931	X
	0.12224	0.13054	1.22401	1.30979	0.02214	0.03063	0.22194	0.30905	0.4945	1.54658
	1.43%	2.15%	0.82%	0.93%	2.76%	4.97%	0.79%	3.69%	6.33%	8.55%
0.4	0.3	0.4	1.5	2.0	0.1	0.2	0.5	1.0	1.3125	X
	0.2998	0.40095	1.49855	1.99848	0.09978	0.20087	0.49807	0.99834	1.32128	9.16262
	1.93%	1.89%	0.61%	0.85%	2.92%	3.67%	0.81%	2.35%	3.0%	7.34%
0.6	0.55714	1.13571	1.85714	3.78571	0.25714	0.83571	0.85714	2.78571	2.76385	X
	0.55606	1.12351	1.85354	3.76388	0.25627	0.82439	0.85329	2.76932	2.77179	59.26071
	0.44%	1.91%	1.02%	1.92%	1.1%	2.21%	1.15%	2.64%	8.99%	5.89%
0.8	0.93333	4.13333	2.33333	10.33333	0.53333	3.73333	1.33333	9.33333	5.62963	X
	0.93431	4.14681	2.33523	10.44639	0.5341	3.74652	1.33541	9.44607	5.61005	790.73029
	0.28%	1.18%	0.53%	2.4%	0.42%	1.29%	0.83%	2.64%	2.67%	9.96%
0.9	1.18636	11.12727	2.63636	24.72727	0.73636	10.67727	1.63636	23.72727	8.14125	X
	1.18593	11.11241	2.63293	24.5574	0.73595	10.66243	1.63331	23.55766	8.12877	7429.88892
	0.15%	0.96%	0.29%	2.18%	0.21%	0.99%	0.43%	2.27%	1.55%	9.95%

Figura 4.2: Tabela com os valores para a política de atendimento Last Come First Served.

O número de clientes avaliados em cada valor de utilização é explicitado na seção 1.7.

O número de rodadas e tamanho da fase transiente para cada tipo de experimento é mostrado abaixo (os tamanho das fases transientes também são mostrados no capítulo 3, mas são expostos aqui também para facilidade de leitura):

- $\rho = 0.2$ - F.C.F.S : 5 rodadas - 30.000 clientes na fase transiente.
- $\rho = 0.2$ - L.C.F.S : 4 rodadas - 30.000 clientes na fase transiente.
- $\rho = 0.4$ - F.C.F.S : 11 rodadas - 40.000 clientes na fase transiente.
- $\rho = 0.4$ - L.C.F.S : 5 rodadas - 40.000 clientes na fase transiente.
- $\rho = 0.6$ - F.C.F.S : 14 rodadas - 80.000 clientes na fase transiente.
- $\rho = 0.6$ - L.C.F.S : 4 rodadas - 80.000 clientes na fase transiente.
- $\rho = 0.8$ - F.C.F.S : 19 rodadas - 400.000 clientes na fase transiente.
- $\rho = 0.8$ - L.C.F.S : 31 rodadas - 400.000 clientes na fase transiente.
- $\rho = 0.9$ - F.C.F.S : 76 rodadas - 500.000 clientes na fase transiente.
- $\rho = 0.9$ - L.C.F.S : 106 rodadas - 500.000 clientes na fase transiente.

Como pode ser visto nas tabelas com os resultados, todos os valores analíticos se encontram dentro do intervalo de confiança estipulado pelo simulador.

4.2 Comentários

O valor da variância do tempo de espera da fila 2 não pode ser verificado analiticamente, portanto não há como ter certeza se o seu valor real se encontra dentro do intervalo de confiança. Porém podemos afirmar que há uma probabilidade grande dele se encontrar dentro do intervalo pois todos os outros valores estimados para a fila 2 estão dentro dos intervalos de confiança calculados.

Esse mesmo valor foi o que comumente mais demorou a convergir para o intervalo de confiança válido, como mostrado nas tabelas 4.1 e 4.2. Por isso usamos ele como métrica para definir a fase transiente como está mostrado na seção 3.

Verificamos empiricamente que avaliar 100.000 clientes por rodada de simulação fez com que os casos mais triviais, como por exemplo valor de utilização 0,2 e política de atendimento F.C.F.S chegassem aos valores desejados em um número menor de rodadas; e que os casos mais críticos, como $\rho = 0.9$ e política L.C.F.S, convergissem ao resultado de maneira adequada.

Ao executar o simulador, para os diversos casos, verificamos o fato de que mesmo com o acréscimo de rodadas, o intervalo de confiança pode aumentar. Fato este que é explicado em que certas rodadas podem gerar médias relativamente distantes umas das outras, aumentando assim o desvio padrão numa taxa maior que a raiz quadrada do número de amostras. Nos resultados isso pôde ser verificado nos casos $\rho = 0.4$ com política F.C.F.S e $\rho = 0.6$ com política F.C.F.S, onde o número de rodadas necessário para se chegar a um resultado válido é bem maior do que casos bastante parecidos, como $\rho = 0.4$ com política L.C.F.S.

Nos casos onde o valor de utilização se aproxima do limite para o sistema entrar em gargalo($\rho = 0.8$ e $\rho = 0.9$), o valor das variâncias encontradas diferem significativamente entre as duas políticas de atendimento usadas. No caso da fila 2, a variância para a política L.C.F.S chega a ser aproximadamente 10 vezes maior que a variância para a política F.C.F.S, em ambos os valores de utilização.

O caso mais crítico que foi avaliado ($\rho = 0.9$ e política L.C.F.S), requeriu um tempo muito maior para convergir ao resultado do que os demais casos, acreditamos que isso se deve a seu valor de utilização estar bem próximo do valor em que o sistema entra em gargalo, e que a variância da fila 2 é de uma ordem de grandeza muito maior que todos os demais valores calculados, portanto demora mais para convergir o seu intervalo de confiança a um resultado válido.

Capítulo 5

Otimização

O fator mínimo que satisfaz a validação do intervalo de confiança para todos os valores de utilização é o mesmo que satisfaz o caso mais crítico, ou seja, com o sistema com valor de utilização igual a 0,9.

Como o método utilizado para a simulação foi o replicativo, o tamanho da fase transiente é considerado em cada rodada do simulador.

5.1 Fatores mínimos:

O valor calculado para os fatores mínimos de cada política são mostrados a seguir:

F.C.F.S (First Come First Served):

- FATOR MÍNIMO = $(\#rodadas)*(tamanhodarodada+fasestransientes)$
- FATOR MÍNIMO = $76 * (100.000 + 500.000)$
- FATOR MÍNIMO = 45.600.000

L.C.F.S (Last Come First Served):

- FATOR MÍNIMO = $(\#rodadas)*(tamanhodarodada+fasestransientes)$
- FATOR MÍNIMO = $106 * (100.000 + 500.000)$
- FATOR MÍNIMO = 63.600.000

Capítulo 6

Conclusões

Coloque aqui seus comentários finais. Descreva dificuldades encontradas, as otimizações feitas, e outras conclusões que você tirou do trabalho. Comente o que poderia ser melhorado, como, por exemplo, o tempo de execução do seu programa. Adicione quaisquer comentários que você julgar relevante. Cada uma das seções terá sua avaliação. Portanto, não deixe de colocar nenhuma seção no seu relatório. Se você não incluir uma seção, deixe-a em branco, mas não altere a numeração. Recomendamos fortemente que isso não ocorra. Não deixe de ler o capítulo de simulação na apostila.

Capítulo 7

Implementação

Este capítulo conterá o código fonte do simulador, dividido por tipo de módulo e ordenados por importância.

7.1 Classes:

7.1.1 Simulator:

Classe que implementa a lógica principal do simulador, processa as rodadas tratando os eventos e as chegadas dos clientes. E calcula as estimativas das variáveis aleatórias.

```
import sys
from collections import deque
from util.constants import *
from util.progress_bar import ProgressBar
from util import estimator as est
from util import dist
from client import *
from event_heap import *


class Simulator:

    # Inicialização do simulador
    def __init__(self, entry_rate, warm_up, service_policy, clients, server_rate
```

```
        # Número total de clientes = fase transiente + clientes a serem avaliado
        self.total_clients = warm_up + clients
        self.samples = 1
        self.server_rate = server_rate
        self.entry_rate = entry_rate
        self.warm_up = warm_up
        # Definido aqui o método a ser utilizado para retirar os clientes da fil
        # dependendo da política de atendimento usada.
        if service_policy == FCFS:
            Simulator.__dict__['pop_queue1'] = Simulator.pop_queue1_fcfs
            Simulator.__dict__['pop_queue2'] = Simulator.pop_queue2_fcfs
            self.service_policy = 'First Come First Served (FCFS)'
        elif service_policy == LCFS:
            Simulator.__dict__['pop_queue1'] = Simulator.pop_queue1_lcfs
            Simulator.__dict__['pop_queue2'] = Simulator.pop_queue2_lcfs
            self.service_policy = 'Last Come First Served (LCFS)'
        self.init_sample()
        # Define o dicionário que irá guardar a soma e a soma dos quadrados das
        self.sums = { 'm_s_W1': 0, 'm_s_s_W1': 0, 'v_s_W1': 0, 'v_s_s_W1': 0,
                      'm_s_N1': 0, 'm_s_s_N1': 0, 'm_s_Nq1': 0, 'm_s_s_Nq1': 0,
                      'm_s_T1': 0, 'm_s_s_T1': 0, 'm_s_W2': 0, 'm_s_s_W2': 0,
                      'v_s_W2': 0, 'v_s_s_W2': 0, 'm_s_N2': 0, 'm_s_s_N2': 0,
                      'm_s_Nq2': 0, 'm_s_s_Nq2': 0, 'm_s_T2': 0, 'm_s_s_T2': 0 }
        # Dicionário que irá guardar os resultados de cada estimador calculados
        self.results = {}

    # Inicializa as estruturas de dados para cada rodada
    def init_sample(self):
        # Filas do sistema.
        self.queue1 = deque([])
        self.queue2 = deque([])
        # Cliente que está no servidor ( Quando esta variável for nula significa
        self.server_current_client = None
        # Lista dos clientes que entraram no sistema durante a rodada.
        self.clients = []
        # Dicionário com a soma das variáveis que indicam o número de pessoas na
        self.N_samples = { 'Nq_1': 0, 'N_1': 0, 'Nq_2': 0, 'N_2': 0 }
        self.warm_up_sample = self.warm_up
        # Tempo do simulador.
        self.t = 0.0
        # Tempo do evento anterior ao que está sendo processado.
```

```
        self.previous_event_time = 0.0
        # Lista de eventos.
        self.events = EventHeap()
        # Inicializa o simulador com o evento de chegada do primeiro cliente ao
        self.events.push((dist.exp_time(self.entry_rate), INCOMING))

    # Inicia o simulador
    def start(self):
        # Inicializa a barra usada para medir o progresso do simulador
        # Ela é contabilizada de acordo com o valor do menor intervalo de confia
        # Chegando a 100% quando o intervalo chega a 10% da média do estimador
        prog = ProgressBar(0, 0.9, 77, mode='fixed', char='#')
        print "Processando as rodadas:"
        print prog, '\r',
        sys.stdout.flush()

        # Loop principal do simulador.
        # Termina quando todos os intervalos de confiança forem menores que 10%
        while not(self.valid_confidence_interval()):
            # Loop de cada rodada, processa um evento a cada iteração.
            while len(self.clients) < self.total_clients:
                self.process_event()
            self.discard_clients()
            # Processa os dados gerados por uma rodada.
            self.process_sample()
            if self.samples > 1:
                self.calc_results()
                prog.update_amount(max(prog.amount, self.pb_amount()))
                print prog, '\r',
            self.samples += 1
            sys.stdout.flush()
        print

    # Método que processa um evento
    def process_event(self):
        # Remove um evento da lista para ser processado e atualiza o tempo do si
        self.t, event_type = self.events.pop()

        # Evento do tipo: Chegada ao sistema.
        if event_type == INCOMING:
            self.update_n()
```

```
        # Define a cor do cliente, verificando se ele chegou durante a fase
        if self.warm_up_sample > 0:
            new_client = Client(TRANSIENT)
            self.warm_up_sample -= 1
        else:
            new_client = Client(EQUILIBRIUM)
        # Adiciona o cliente na fila 1 e define o seu tempo de chegada nessa
        new_client.set_queue(1)
        new_client.set_arrival(self.t)
        self.queue1.append(new_client)
        self.clients.append(new_client)
        # Assim que uma chegada é processada, adiciona outro evento de chega
        self.events.push((self.t + dist.exp_time(self.entry_rate), INCOMING))
        # Se o servidor estiver ocioso, adiciona o evento Entrada ao servido
        if not self.server_current_client:
            self.events.push((self.t, SERVER_1_IN))

    # Evento do tipo: Entrada ao servidor pela fila 1.
    elif event_type == SERVER_1_IN:
        # Define o tempo que o cliente vai ficar no servidor.
        server_time = dist.exp_time(self.server_rate)
        # Adiciona o cliente no servidor e define o seu tempo de saída da fi
        self.server_current_client = self.pop_queue1()
        self.server_current_client.set_leave(self.t)
        self.server_current_client.set_server(server_time)
        # Adiciona o evento Saída do servidor na lista.
        self.events.push((self.t + server_time, SERVER_OUT))

    # Evento do tipo: Entrada ao servidor pela fila 2.
    elif event_type == SERVER_2_IN:
        # Define o tempo que o cliente vai ficar no servidor.
        server_time = dist.exp_time(self.server_rate)
        # Adiciona o cliente no servidor e define o seu tempo de saída da fi
        self.server_current_client = self.pop_queue2()
        self.server_current_client.set_leave(self.t)
        self.server_current_client.set_server(server_time)
        # Adiciona o evento Saída do servidor na lista.
        self.events.push((self.t + server_time, SERVER_OUT))

    # Evento do tipo: Saída do servidor.
    elif event_type == SERVER_OUT:
```

```
            self.update_n()
            # Se a fila 1 possuir clientes, adiciona o evento Entrada ao servido
            if self.queue1:
                self.events.push((self.t, SERVER_1_IN))
            # Se a fila 2 possuir clientes e a fila 1 vazia, ou se o sistema est
            # está no servidor entrou nele pela fila 1, adiciona o evento Entrad
            elif self.queue2 or self.server_current_client.queue == 1:
                self.events.push((self.t, SERVER_2_IN))

            # Se o cliente que está no servidor entrou nele pela fila 1, adicion
            if self.server_current_client.queue == 1:
                self.queue_2_in()
            # Senão, define que ele foi servido e saiu do sistema.
            else:
                self.server_current_client.set_served(1)
            self.server_current_client = None

    # Método que trata a entrada de um cliente na fila 2. Encapsulado para melho
    def queue_2_in(self):
        # Pega o cliente do servidor e o adiciona na fila 2, definindo seu tempo
        client = self.server_current_client
        self.queue2.append(client)
        client.set_queue(2)
        client.set_arrival(self.t)

    # Atualiza o número de pessoas nas filas a cada chegada, é chamado no início
    # fazem o tempo do simulador passar (Chegada ao sistema e Saída do servidor)
    def update_n(self):
        # Calcula o intervalo de tempo entre o evento atual e o imediatamente an
        delta = self.t - self.previous_event_time
        # Define o número de pessoas nas filas de espera
        n1 = len(self.queue1)
        n2 = len(self.queue2)
        # Soma às variáveis estimadas (Nq1) e (Nq2) o número de clientes na fila
        # intervalo de tempo (delta) em que as filas ficaram com esse número de
        self.N_samples['Nq_1'] += n1*delta
        self.N_samples['Nq_2'] += n2*delta
        # Testa se o cliente que está no servidor, se ele estiver ocupado, veio
        if self.server_current_client:
            if self.server_current_client.queue == 1:
                n1 += 1
```

```
            elif self.server_current_client.queue == 2:
                n2 += 1
        # Soma às variáveis estimadas (N1) e (N2) o número de clientes na fila m
        # intervalo de tempo (delta) em que as filas ficaram com esse número de
        self.N_samples['N_1'] += n1*delta
        self.N_samples['N_2'] += n2*delta
        # Atualiza o valor do tempo do evento anterior pelo evento atual, já que
        self.previous_event_time = self.t

    # Método que descarta os clientes da fase transiente e os clientes que ainda
    # da rodada.
    def discard_clients(self):
        served_clients = []
        for client in self.clients:
            if client.served and client.color == EQUILIBRIUM:
                served_clients.append(client)
        self.clients = served_clients

    # Método que processa os dados gerados por uma rodada.
    def process_sample(self):
        s_wait_1 = 0; s_s_wait_1 = 0
        s_wait_2 = 0; s_s_wait_2 = 0
        s_server_1 = 0; s_server_2 = 0

        # Loop que faz a soma e a soma dos quadrados dos tempos de espera e a so
        # Dos clientes na fila 1 e na fila 2.
        for client in self.clients:
            s_wait_1 += client.wait(1)
            s_s_wait_1 += client.wait(1)**2
            s_server_1 += client.server[1]
            s_wait_2 += client.wait(2)
            s_s_wait_2 += client.wait(2)**2
            s_server_2 += client.server[2]

        # Adiciona à soma e à soma dos quadrados dos estimadores os valores esti
        self.sums['m_s_W1'] += est.mean(s_wait_1, len(self.clients))
        self.sums['m_s_s_W1'] += est.mean(s_wait_1, len(self.clients))**2
        self.sums['v_s_W1'] += est.variance(s_wait_1, s_s_wait_1, len(self.clien
        self.sums['v_s_s_W1'] += est.variance(s_wait_1, s_s_wait_1, len(self.cli
        self.sums['m_s_N1'] += est.mean(self.N_samples['N_1'], self.t)
        self.sums['m_s_s_N1'] += est.mean(self.N_samples['N_1'], self.t)**2
```

```
    self.sums['m_s_Nq1'] += est.mean(self.N_samples['Nq_1'], self.t)
    self.sums['m_s_s_Nq1'] += est.mean(self.N_samples['Nq_1'], self.t)**2
    self.sums['m_s_T1'] += est.mean(s_wait_1, len(self.clients)) + est.mean(
    self.sums['m_s_s_T1'] += (est.mean(s_wait_1, len(self.clients)) + est.me
    self.sums['m_s_W2'] += est.mean(s_wait_2, len(self.clients))
    self.sums['m_s_s_W2'] += est.mean(s_wait_2, len(self.clients))**2
    self.sums['v_s_W2'] += est.variance(s_wait_2, s_s_wait_2, len(self.clien
    self.sums['v_s_s_W2'] += est.variance(s_wait_2, s_s_wait_2, len(self.cli
    self.sums['m_s_N2'] += est.mean(self.N_samples['N_2'], self.t)
    self.sums['m_s_s_N2'] += est.mean(self.N_samples['N_2'], self.t)**2
    self.sums['m_s_Nq2'] += est.mean(self.N_samples['Nq_2'], self.t)
    self.sums['m_s_s_Nq2'] += est.mean(self.N_samples['Nq_2'], self.t)**2
    self.sums['m_s_T2'] += est.mean(s_wait_2, len(self.clients)) + est.mean(
    self.sums['m_s_s_T2'] += (est.mean(s_wait_2, len(self.clients)) + est.me
    # Inicializa as estruturas de dados para a próxima rodada.
    self.init_sample()

# Método que calcula os resultados (valor e intervalo de confiança) para cad
def calc_results(self):
    self.results = {
        'E[N1]'  : { 'value' : est.mean(self.sums['m_s_N1'], self.samples),
        'E[N2]'  : { 'value' : est.mean(self.sums['m_s_N2'], self.samples),
        'E[T1]'  : { 'value' : est.mean(self.sums['m_s_T1'], self.samples),
        'E[T2]'  : { 'value' : est.mean(self.sums['m_s_T2'], self.samples),
        'E[Nq1]' : { 'value' : est.mean(self.sums['m_s_Nq1'], self.samples),
        'E[Nq2]' : { 'value' : est.mean(self.sums['m_s_Nq2'], self.samples),
        'E[W1]'  : { 'value' : est.mean(self.sums['m_s_W1'], self.samples),
        'E[W2]'  : { 'value' : est.mean(self.sums['m_s_W2'], self.samples),
        'V(W1)'  : { 'value' : est.mean(self.sums['v_s_W1'], self.samples),
        'V(W2)'  : { 'value' : est.mean(self.sums['v_s_W2'], self.samples),
    }

# Método que atualiza a barra de progresso com o valor do menor intervalo de
def pb_amount(self):
    return 1 - max((2.0*self.results['E[N1]']['c_i']/self.results['E[N1]']['
                   (2.0*self.results['E[N2]']['c_i']/self.results['E[N2]']['
                   (2.0*self.results['E[T1]']['c_i']/self.results['E[T1]']['
                   (2.0*self.results['E[T2]']['c_i']/self.results['E[T2]']['
                   (2.0*self.results['E[Nq1]']['c_i']/self.results['E[Nq1]']
                   (2.0*self.results['E[Nq2]']['c_i']/self.results['E[Nq2]']
                   (2.0*self.results['E[W1]']['c_i']/self.results['E[W1]']['
```

```
                    (2.0*self.results['E[W2]']['c_i']/self.results['E[W2]']['
                    (2.0*self.results['V(W1)']['c_i']/self.results['V(W1)']['
                    (2.0*self.results['V(W2)']['c_i']/self.results['V(W2)']['

# Método que testa se todos os intervalos de confiança são válidos.
# Só faz a validação a partir da terceira rodada.
def valid_confidence_interval(self):
    return not(self.samples <= 2) and \
           (2.0*self.results['E[N1]']['c_i']  <= 0.1*self.results['E[N1]']['
           (2.0*self.results['E[N2]']['c_i']  <= 0.1*self.results['E[N2]']['
           (2.0*self.results['E[T1]']['c_i']  <= 0.1*self.results['E[T1]']['
           (2.0*self.results['E[T2]']['c_i']  <= 0.1*self.results['E[T2]']['
           (2.0*self.results['E[Nq1]']['c_i'] <= 0.1*self.results['E[Nq1]']['
           (2.0*self.results['E[Nq2]']['c_i'] <= 0.1*self.results['E[Nq2]']['
           (2.0*self.results['E[W1]']['c_i']  <= 0.1*self.results['E[W1]']['
           (2.0*self.results['E[W2]']['c_i']  <= 0.1*self.results['E[W2]']['
           (2.0*self.results['V(W1)']['c_i']  <= 0.1*self.results['V(W1)']['
           (2.0*self.results['V(W2)']['c_i']  <= 0.1*self.results['V(W2)']['

# Método que exibe os resultados junto com o número de rodadas processadas e
def report(self):
    print "Exibindo os resultados:"
    for key in self.results.keys():
        print key, ': ', self.results[key]['value'], ' - I.C: ', self.result
    print "Número de rodadas :", self.samples
    return self.results

# Métodos que tratam o trânsito dos clientes das filas para o servidor, de a
@staticmethod
def pop_queue1_fcfs(instance):
    return instance.queue1.popleft()

@staticmethod
def pop_queue2_fcfs(instance):
    return instance.queue2.popleft()

@staticmethod
def pop_queue1_lcfs(instance):
    return instance.queue1.pop()

@staticmethod
```

```
    def pop_queue2_lcfs(instance):
        return instance.queue2.pop()
```

7.2 Módulos utilitários:

bla