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

Para testar a correção do simulador iremos acompanhar 10 clientes aleatórios nos quatro tipos iniciais de experimento. Se todos esses clientes gerarem a ordem esperada de eventos (Chegada ao sistema; Entrada ao servidor pela fila 1; Saída do servidor; Entrada ao servidor pela fila 2 e Saída do servidor) e eles passarem respectivamente pela fila 1, servidor, fila 2 e servidor, o simulador estará tratando os clientes da forma correta, portanto, estará gerando os valores corretos para os estimadores.

A rotina de testes de correção é mostrada na seção 7.4.

O resultado de um dos testes feitos para o caso do valor de utilização ser $\rho = 0,4$ e política de atendimento F.C.F.S está exposto abaixo:

```
teste com taxa 0.2 , tamanho de fase transiente igual a 40000 , 100000 clientes
avaliados e política de atendimento F.C.F.S (First Come First Served):

Cliente 29003 gerou o evento Chegada ao sistema.
Cliente 29003 entrou na fila 1.
Cliente 29003 gerou o evento Entrada ao servidor pela fila 1.
Cliente 29003 entrou no servidor.
Cliente 29003 gerou o evento Saída do servidor.
Cliente 29003 entrou na fila 2.
Cliente 29003 gerou o evento Entrada ao servidor pela fila 2.
Cliente 29003 entrou no servidor.
Cliente 29003 gerou o evento Saída do servidor.
Cliente 39628 gerou o evento Chegada ao sistema.
Cliente 39628 entrou na fila 1.
Cliente 39628 gerou o evento Entrada ao servidor pela fila 1.
Cliente 39628 entrou no servidor.
Cliente 39628 gerou o evento Saída do servidor.
```

```
Cliente 39628 entrou na fila 2.
Cliente 39628 gerou o evento Entrada ao servidor pela fila 2.
Cliente 39628 entrou no servidor.
Cliente 39628 gerou o evento Saída do servidor.
Cliente 41689 gerou o evento Chegada ao sistema.
Cliente 41689 entrou na fila 1.
Cliente 41689 gerou o evento Entrada ao servidor pela fila 1.
Cliente 41689 entrou no servidor.
Cliente 41689 gerou o evento Saída do servidor.
Cliente 41689 entrou na fila 2.
Cliente 41689 gerou o evento Entrada ao servidor pela fila 2.
Cliente 41689 entrou no servidor.
Cliente 41689 gerou o evento Saída do servidor.
Cliente 41748 gerou o evento Chegada ao sistema.
Cliente 41748 entrou na fila 1.
Cliente 41748 gerou o evento Entrada ao servidor pela fila 1.
Cliente 41748 entrou no servidor.
Cliente 41748 gerou o evento Saída do servidor.
Cliente 41748 entrou na fila 2.
Cliente 41748 gerou o evento Entrada ao servidor pela fila 2.
Cliente 41748 entrou no servidor.
Cliente 41748 gerou o evento Saída do servidor.
Cliente 45957 gerou o evento Chegada ao sistema.
Cliente 45957 entrou na fila 1.
Cliente 45957 gerou o evento Entrada ao servidor pela fila 1.
Cliente 45957 entrou no servidor.
Cliente 45957 gerou o evento Saída do servidor.
Cliente 45957 entrou na fila 2.
Cliente 45957 gerou o evento Entrada ao servidor pela fila 2.
Cliente 45957 entrou no servidor.
Cliente 45957 gerou o evento Saída do servidor.
Cliente 63379 gerou o evento Chegada ao sistema.
Cliente 63379 entrou na fila 1.
Cliente 63379 gerou o evento Entrada ao servidor pela fila 1.
Cliente 63379 entrou no servidor.
Cliente 63379 gerou o evento Saída do servidor.
Cliente 63379 entrou na fila 2.
Cliente 63379 gerou o evento Entrada ao servidor pela fila 2.
Cliente 63379 entrou no servidor.
Cliente 63379 gerou o evento Saída do servidor.
Cliente 65525 gerou o evento Chegada ao sistema.
```

```
Cliente 65525 entrou na fila 1.
Cliente 65525 gerou o evento Entrada ao servidor pela fila 1.
Cliente 65525 entrou no servidor.
Cliente 65525 gerou o evento Saída do servidor.
Cliente 65525 entrou na fila 2.
Cliente 65525 gerou o evento Entrada ao servidor pela fila 2.
Cliente 65525 entrou no servidor.
Cliente 65525 gerou o evento Saída do servidor.
Cliente 74955 gerou o evento Chegada ao sistema.
Cliente 74955 entrou na fila 1.
Cliente 74955 gerou o evento Entrada ao servidor pela fila 1.
Cliente 74955 entrou no servidor.
Cliente 74955 gerou o evento Saída do servidor.
Cliente 74955 entrou na fila 2.
Cliente 74955 gerou o evento Entrada ao servidor pela fila 2.
Cliente 74955 entrou no servidor.
Cliente 74955 gerou o evento Saída do servidor.
Cliente 84645 gerou o evento Chegada ao sistema.
Cliente 84645 entrou na fila 1.
Cliente 84645 gerou o evento Entrada ao servidor pela fila 1.
Cliente 84645 entrou no servidor.
Cliente 84645 gerou o evento Saída do servidor.
Cliente 84645 entrou na fila 2.
Cliente 84645 gerou o evento Entrada ao servidor pela fila 2.
Cliente 84645 entrou no servidor.
Cliente 84645 gerou o evento Saída do servidor.
Cliente 108589 gerou o evento Chegada ao sistema.
Cliente 108589 entrou na fila 1.
Cliente 108589 gerou o evento Entrada ao servidor pela fila 1.
Cliente 108589 entrou no servidor.
Cliente 108589 gerou o evento Saída do servidor.
Cliente 108589 entrou na fila 2.
Cliente 108589 gerou o evento Entrada ao servidor pela fila 2.
Cliente 108589 entrou no servidor.
Cliente 108589 gerou o evento Saída do servidor.
```

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.

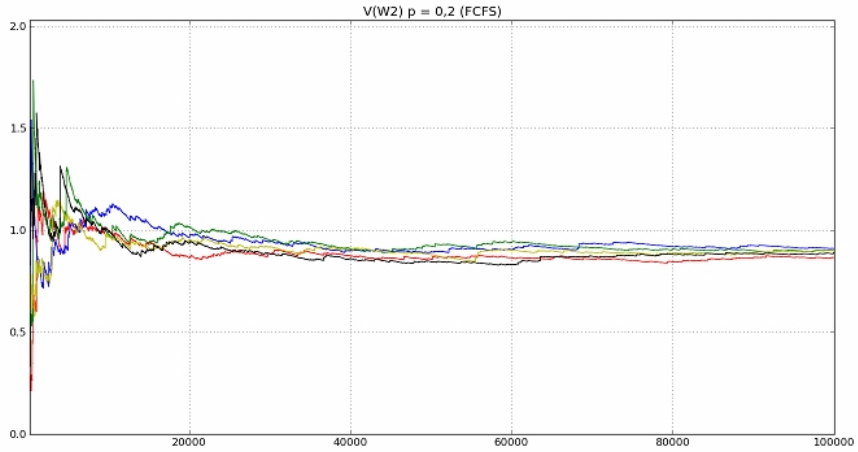
Foram gerados diversos gráficos para cada utilização, e em cada um deles a semente é diferente, já que ela é definida como está exposto na seção 1.4. Todos mostraram o mesmo comportamento. Escolhemos um de cada tipo de experimento para mostrar no relatório.

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.

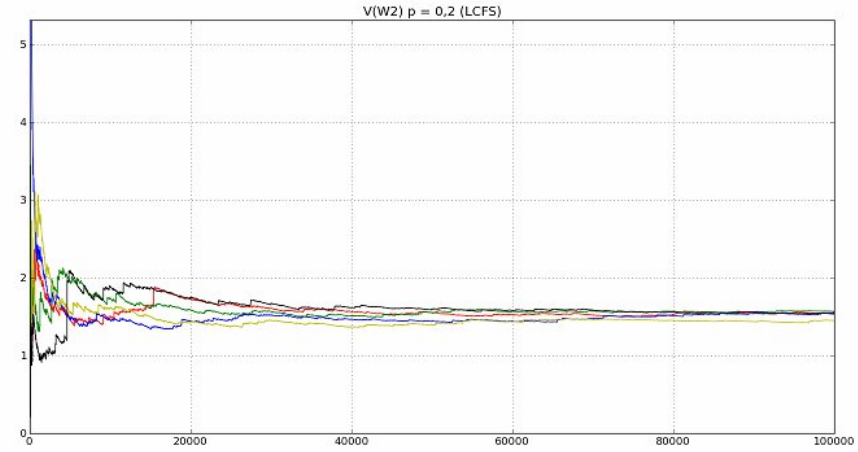
3.1 No simulador

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

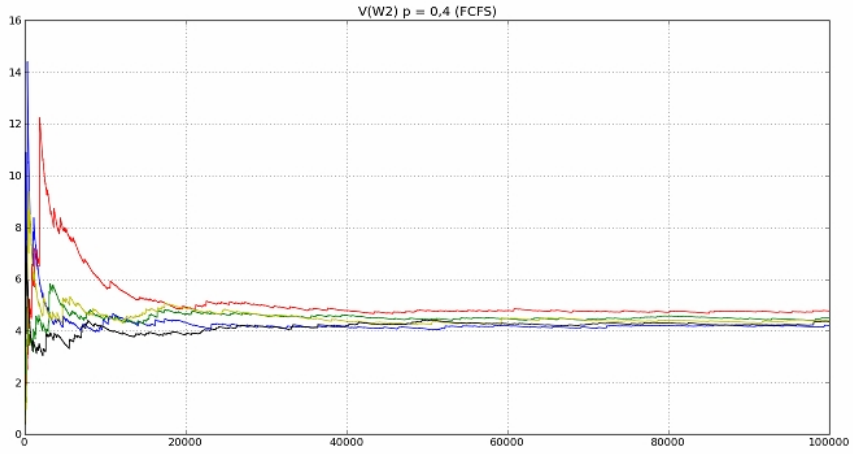


(a) First Come First Served

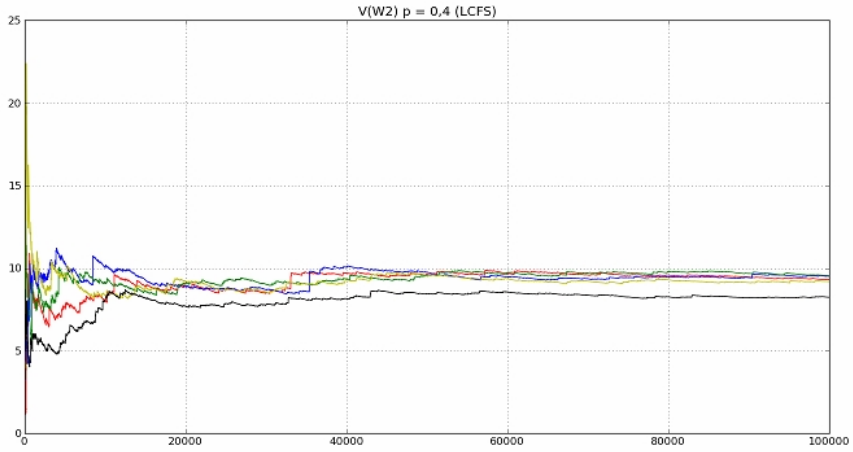


(b) Last Come First Served

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.

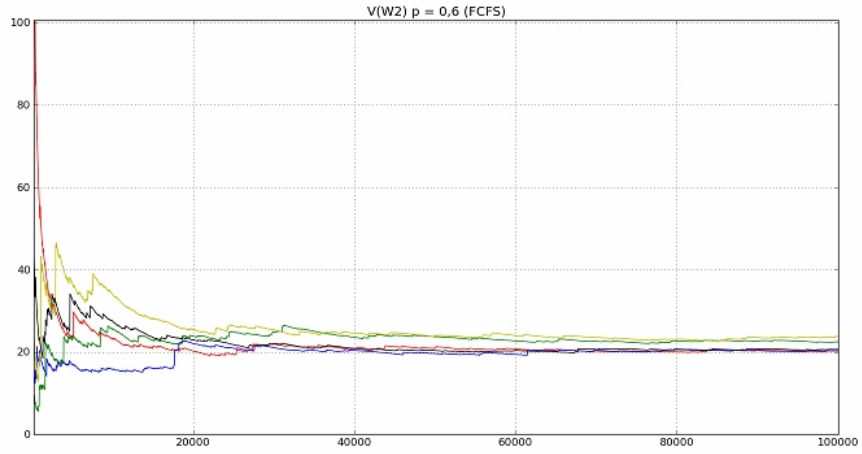


(a) First Come First Served

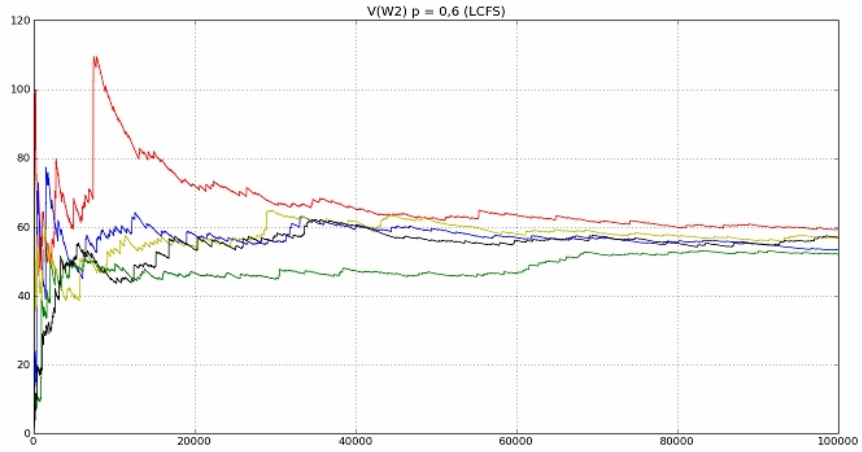


(b) Last Come First Served

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.

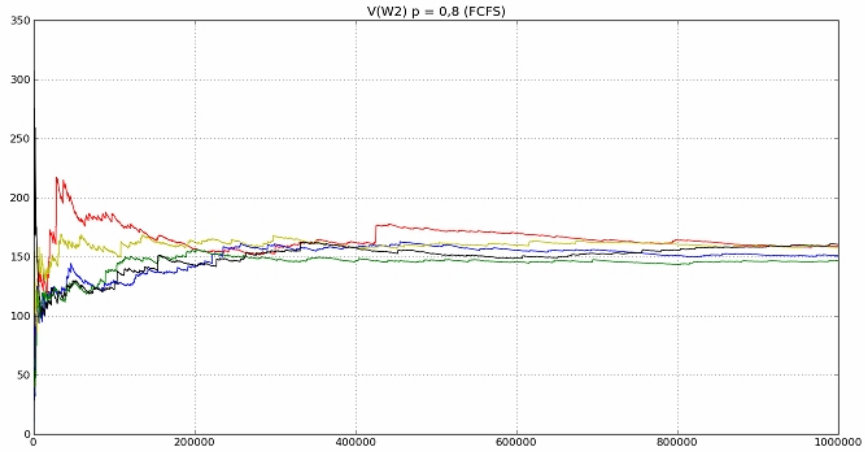


(a) First Come First Served

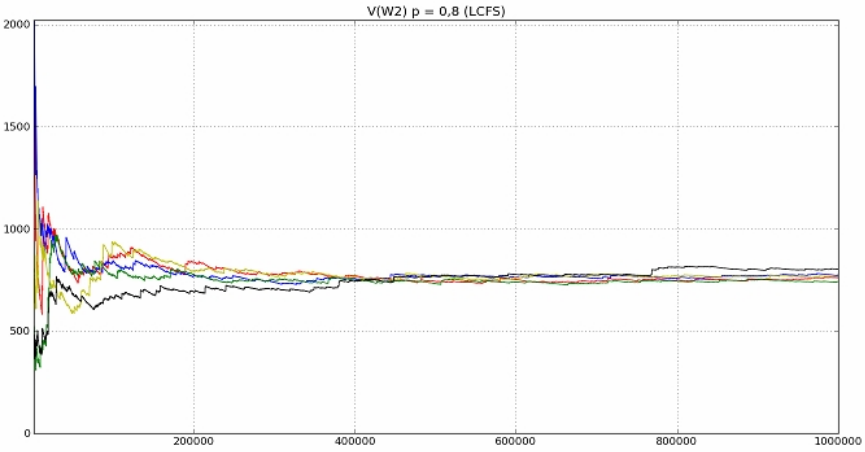


(b) Last Come First Served

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.

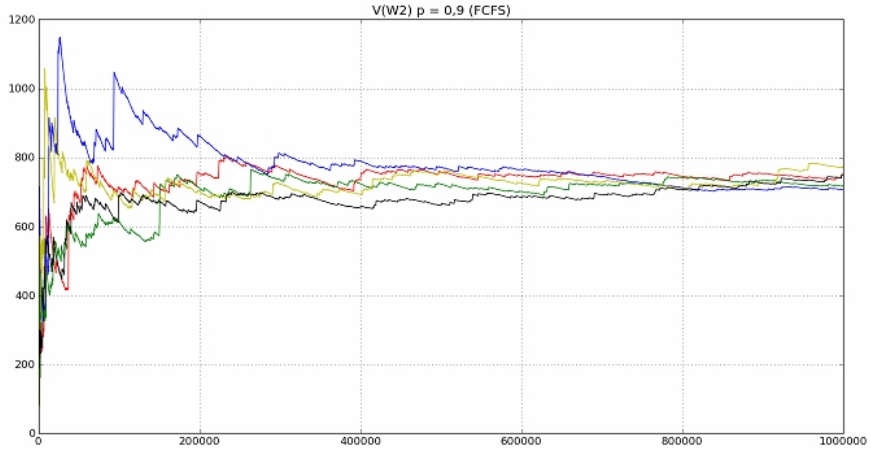


(a) First Come First Served

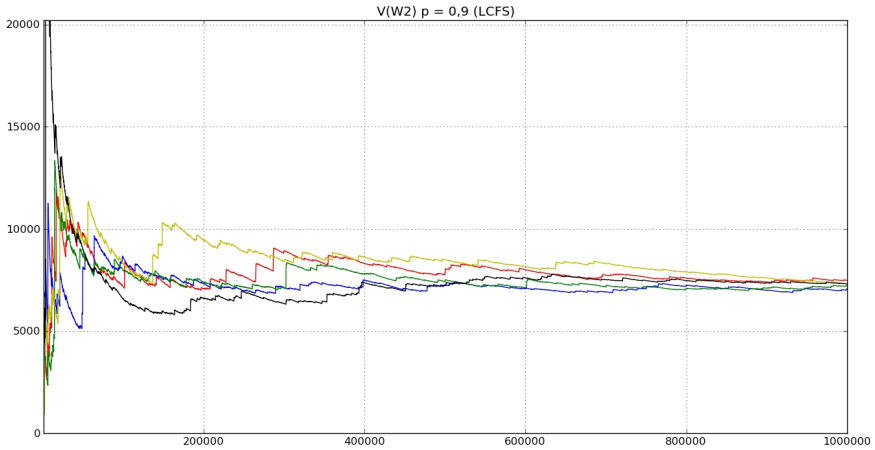


(b) Last Come First Served

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

Inicialmente encontramos dificuldades em implementar a lógica do simulador em si. Seguindo as instruções do capítulo de simulação da apostila, conseguimos ter uma ideia razoável das estruturas de dados necessárias e do fluxo desses dados pelo simulador para que ele gere os resultados corretos. Por exemplo, a lista de eventos, a estrutura de cada evento e o modo com que esses eventos são tratados e gerados pelo simulador são implementados da mesma forma que é explicada na apostila.

A forma como implementamos a lista de eventos do simulador mudou duas vezes durante a codificação do trabalho, já que encontramos dificuldades em encontrar a forma mais adequada em termos de perfomance para criar e gerenciar essa lista. Tendo em vista que ela é a estrutura mais importante do simulador, onde a maior parte do processamento é feito em cima dela, vimos como necessidade essencial otimiza-la da melhor maneira possível.

O uso da linguagem python facilitou bastante a implementação dos cálculos estatísticos do simulador, utilizando módulos como o scipy para o cálculo do valor da distribuição t de student para qualquer número de amostras. Esta facilidade pode ser comprovada analisando o tamanho dos métodos usados para o cálculo das médias, variâncias e intervalos de confiança dos módulos utilitários. O gerador de números aleatórios do python também facilitou bastante a implementação, pois, como é explicado na seção 1.4, não foi necessário se preocupar com o valor da semente do gerador a cada rodada do método replicativo.

A geração dos gráficos necessários para a estimativa da fase transiente também foi feita sem menores dificuldades, devido ao uso da linguagem python.

Usamos o módulo psyco para agilizar a execução do programa.

A implementação desse simulador aumentou significativamente o nosso entendimento sobre o sistema de rede de filas, foi possível verificar na prática

as diferenças entre um sistema transiente e em equilíbrio; a evolução de um cliente dentro do sistema; o fato de que as médias são iguais para as duas políticas de atendimento, porém as variâncias são bastante distintas; e que os valores calculados de forma analítica puderam ser comprovados executando o simulador para todos os casos.

Capítulo 7

Listagem documentada do programa

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.

```
1 import sys, random, math
2 from collections import deque
3 from util.constants import *
4 from util.progress_bar import ProgressBar
5 from util import estimator as est
6 from util import dist
7 from client import *
8 from event_heap import *
9
10
11 class Simulator:
12
13     # Inicializacao do simulador
```

```
14      def __init__(self, entry_rate, warm_up,
            service_policy, clients, server_rate=1.0, test=
            False):
15          # Numero total de clientes = fase transiente +
                clientes a serem avaliados
16          self.total_clients = warm_up + clients
17          self.samples = 1
18          self.server_rate = server_rate
19          self.entry_rate = entry_rate
20          self.warm_up = warm_up
21          # Definido aqui o metodo a ser utilizado para
                retirar os clientes da fila e coloca-los no
                servidor,
22          # dependendo da politica de atendimento usada.
23          if service_policy == FCFS:
24              Simulator.__dict__['pop_queue1'] =
                    Simulator.pop_queue1_fcfs
25              Simulator.__dict__['pop_queue2'] =
                    Simulator.pop_queue2_fcfs
26              self.service_policy = 'First␣Come␣First␣
                    Served␣(FCFS)'
27          elif service_policy == LCFS:
28              Simulator.__dict__['pop_queue1'] =
                    Simulator.pop_queue1_lcfs
29              Simulator.__dict__['pop_queue2'] =
                    Simulator.pop_queue2_lcfs
30              self.service_policy = 'Last␣Come␣First␣
                    Served␣(LCFS)'
31          self.init_sample()
32          # Define o dicionario que ira guardar a soma e
                a soma dos quadrados das medias e variancias
                 estimadas a cada rodada.
33          self.sums = { 'm_s_W1': 0, 'm_s_s_W1': 0, '
                v_s_W1': 0, 'v_s_s_W1': 0,
34                        'm_s_N1': 0, 'm_s_s_N1': 0, '
                              m_s_Nq1': 0, 'm_s_s_Nq1': 0,
35                        'm_s_T1': 0, 'm_s_s_T1': 0, '
                              m_s_W2': 0, 'm_s_s_W2': 0,
36                        'v_s_W2': 0, 'v_s_s_W2': 0, '
                              m_s_N2': 0, 'm_s_s_N2': 0,
```

```
37                          'm_s_Nq2': 0, 'm_s_s_Nq2': 0, '
                              m_s_T2': 0, 'm_s_s_T2': 0 }
38          # Dicionario que ira guardar os resultados de
                cada estimador calculados pelo simulador.
39          self.results = {}
40          # Define a lista com os clientes que serao
                testados, caso o simulador esteja em modo de
                 teste.
41          self.test = test
42          self.test_list = []
43          if self.test:
44              self.init_test()
45
46      # Inicializa as estruturas de dados para cada
            rodada
47      def init_sample(self):
48          # Filas do sistema.
49          self.queue1 = deque([])
50          self.queue2 = deque([])
51          # Cliente que esta no servidor ( Quando esta
                variavel for nula significa que o servidor
                esta ocioso )
52          self.server_current_client = None
53          # Lista dos clientes que entraram no sistema
                durante a rodada.
54          self.clients = []
55          # Dicionario com a soma das variaveis que
                indicam o numero de pessoas nas filas (N) e
                em espera (Nq)
56          self.N_samples = { 'Nq_1': 0, 'N_1': 0, 'Nq_2':
                 0, 'N_2': 0 }
57          self.warm_up_sample = self.warm_up
58          # Tempo do simulador.
59          self.t = 0.0
60          # Tempo do evento anterior ao que esta sendo
                processado.
61          self.previous_event_time = 0.0
62          # Lista de eventos.
63          self.events = EventHeap()
64          # Inicializa o simulador com o evento de
                chegada do primeiro cliente ao sistema.
```

```
65          self.events.push((dist.exp_time(self.entry_rate
                ), INCOMING))
66
67      # Inicia o simulador
68      def start(self):
69          # Inicializa a barra usada para medir o
                progresso do simulador
70          # Ela e contabilizada de acordo com o valor do
                menor intervalo de confianca encontrado a
                cada rodada,
71          # Chegando a 100% quando o intervalo chega a
                10% da media do estimador
72          # Mostrada quando o simulador nao esta definido
                na forma de teste
73          if not(self.test):
74              prog = ProgressBar(0, 0.9, 77, mode='fixed'
                    , char='#')
75              print "Processando as rodadas:"
76              print prog, '\r',
77              sys.stdout.flush()
78
79          # Loop principal do simulador.
80          # Termina quando todos os intervalos de
                confianca forem menores que 10% da media do
                estimador.
81          while not(self.valid_confidence_interval()):
82              # Loop de cada rodada, processa um evento a
                    cada iteracao.
83              while len(self.clients) <= self.
                    total_clients:
84                  self.process_event()
85              self.discard_clients()
86              # Processa os dados gerados por uma rodada.
87              self.process_sample()
88              if self.samples > 1:
89                  self.calc_results()
90                  prog.update_amount(max(prog.amount,
                        self.pb_amount()))
91                  print prog, '\r',
92              self.samples += 1
93              sys.stdout.flush()
```

```
94              # Linha para forcar o teste a executar
                    apenas 1 rodada.
95              if self.test:
96                  break
97          print
98
99      # Metodo que processa um evento
100     def process_event(self):
101         # Remove um evento da lista para ser processado
                e atualiza o tempo do simulador
102         self.t, event_type = self.events.pop()
103
104         # Evento do tipo: Chegada ao sistema.
105         if event_type == INCOMING:
106             self.update_n()
107             # Define a cor do cliente, verificando se
                    ele chegou durante a fase transiente ou
                    nao.
108             if self.warm_up_sample > 0:
109                 new_client = Client(len(self.clients),
                        TRANSIENT)
110                 self.warm_up_sample -= 1
111             else:
112                 new_client = Client(len(self.clients),
                        EQUILIBRIUM)
113             # Adiciona o cliente na fila 1 e define o
                    seu tempo de chegada nessa fila.
114             new_client.set_queue(1)
115             new_client.set_arrival(self.t)
116             self.queue1.append(new_client)
117             self.clients.append(new_client)
118             # Teste de correcao
119             if self.test and (new_client.id in self.
                    test_list):
120                 print "Cliente", new_client.id, "gerou␣
                        o␣evento␣Chegada␣ao␣sistema."
121                 print "Cliente", new_client.id, "entrou
                        ␣na␣fila␣1."
122             # Assim que uma chegada e processada,
                    adiciona outro evento de chegada, dando
                    o tempo que ela ira ocorrer.
```

```
123             self.events.push((self.t + dist.exp_time(
                    self.entry_rate), INCOMING))
124             # Se o servidor estiver ocioso, adiciona o
                    evento Entrada ao servidor pela fila 1
                    para esse cliente na lista.
125             if not self.server_current_client:
126                 self.events.push((self.t, SERVER_1_IN))
127
128         # Evento do tipo: Entrada ao servidor pela fila
                1.
129         elif event_type == SERVER_1_IN:
130             # Define o tempo que o cliente vai ficar no
                    servidor.
131             server_time = dist.exp_time(self.
                    server_rate)
132             # Adiciona o cliente no servidor e define o
                    seu tempo de saida da fila 1.
133             self.server_current_client = self.
                    pop_queue1()
134             self.server_current_client.set_leave(self.t
                    )
135             self.server_current_client.set_server(
                    server_time)
136             # Teste de correcao
137             if self.test and (self.
                    server_current_client.id in self.
                    test_list):
138                 print "Cliente", self.
                        server_current_client.id, "gerou␣o␣
                        evento␣Entrada␣ao␣servidor␣pela␣fila
                        ␣1."
139                 print "Cliente", self.
                        server_current_client.id, "entrou␣no
                        ␣servidor."
140             # Adiciona o evento Saida do servidor na
                    lista.
141             self.events.push((self.t + server_time,
                    SERVER_OUT))
142
143         # Evento do tipo: Entrada ao servidor pela fila
                2.
```

```
144         elif event_type == SERVER_2_IN:
145             # Define o tempo que o cliente vai ficar no
                    servidor.
146             server_time = dist.exp_time(self.
                    server_rate)
147             # Adiciona o cliente no servidor e define o
                    seu tempo de saida da fila 2.
148             self.server_current_client = self.
                    pop_queue2()
149             self.server_current_client.set_leave(self.t
                    )
150             self.server_current_client.set_server(
                    server_time)
151             # Teste de correcao
152             if self.test and (self.
                    server_current_client.id in self.
                    test_list):
153                 print "Cliente", self.
                        server_current_client.id, "gerou␣o␣
                        evento␣Entrada␣ao␣servidor␣pela␣fila
                        ␣2."
154                 print "Cliente", self.
                        server_current_client.id, "entrou␣no
                        ␣servidor."
155             # Adiciona o evento Saida do servidor na
                    lista.
156             self.events.push((self.t + server_time,
                    SERVER_OUT))
157
158         # Evento do tipo: Saida do servidor.
159         elif event_type == SERVER_OUT:
160             self.update_n()
161             # Se a fila 1 possuir clientes, adiciona o
                    evento Entrada ao servidor pela fila 1
                    na lista.
162             if self.queue1:
163                 self.events.push((self.t, SERVER_1_IN))
164             # Se a fila 2 possuir clientes e a fila 1
                    vazia, ou se o sistema estiver vazio e o
                    cliente que
```

```
165             # esta no servidor entrou nele pela fila 1,
                    adiciona o evento Entrada ao servidor
                   pela fila 2 na lista.
166             elif self.queue2 or self.
                   server_current_client.queue == 1:
167                 self.events.push((self.t, SERVER_2_IN))
168
169             # Teste de correcao
170             if self.test and (self.
                   server_current_client.id in self.
                   test_list):
171                 print "Cliente", self.
                       server_current_client.id, "gerou o
                       evento Saida do servidor."
172
173             # Se o cliente que esta no servidor entrou
                   nele pela fila 1, adiciona ele na fila
                   2.
174             if self.server_current_client.queue == 1:
175                 self.queue_2_in()
176             # Senao, define que ele foi servido e saiu
                   do sistema.
177             else:
178                 self.server_current_client.set_served
                       (1)
179             self.server_current_client = None
180
181     # Metodo que trata a entrada de um cliente na fila
           2. Encapsulado para melhor legibilidade.
182     def queue_2_in(self):
183         # Pega o cliente do servidor e o adiciona na
                fila 2, definindo seu tempo de chegada na
               mesma.
184         client = self.server_current_client
185         self.queue2.append(client)
186         client.set_queue(2)
187         client.set_arrival(self.t)
188         # Teste de correcao
189         if self.test and (client.id in self.test_list):
190             print "Cliente", client.id, "entrou na fila
                   2."
```

```
191
192     # Atualiza o numero de pessoas nas filas a cada
            chegada, e chamado no inicio de eventos que
193     # fazem o tempo do simulador passar (Chegada ao
            sistema e Saida do servidor)
194     def update_n(self):
195         # Calcula o intervalo de tempo entre o evento
                atual e o imediatamente anterior.
196         delta = self.t - self.previous_event_time
197         # Define o numero de pessoas nas filas de
                espera
198         n1 = len(self.queue1)
199         n2 = len(self.queue2)
200         # Soma as variaveis estimadas (Nq1) e (Nq2) o
                numero de clientes na fila de espera
                multiplicado pelo
201         # intervalo de tempo (delta) em que as filas
                ficaram com esse numero de clientes.
202         self.N_samples['Nq_1'] += n1*delta
203         self.N_samples['Nq_2'] += n2*delta
204         # Testa se o cliente que esta no servidor, se
                ele estiver ocupado, veio da fila 1 ou da
                fila 2.
205         if self.server_current_client:
206             if self.server_current_client.queue == 1:
207                 n1 += 1
208             elif self.server_current_client.queue == 2:
209                 n2 += 1
210         # Soma as variaveis estimadas (N1) e (N2) o
                numero de clientes na fila multiplicado pelo
211         # intervalo de tempo (delta) em que as filas
                ficaram com esse numero de clientes.
212         self.N_samples['N_1'] += n1*delta
213         self.N_samples['N_2'] += n2*delta
214         # Atualiza o valor do tempo do evento anterior
                pelo evento atual, ja que o simulador vai
                processar o proximo evento.
215         self.previous_event_time = self.t
216
217     # Metodo que descarta os clientes da fase
            transiente e os clientes que ainda estao no
```

```
        sistema apos o termino do processamento
218     # da rodada.
219     def discard_clients(self):
220         served_clients = []
221         for client in self.clients:
222             if client.served and client.color ==
                EQUILIBRIUM:
223                 served_clients.append(client)
224         self.clients = served_clients
225
226     # Metodo que processa os dados gerados por uma
            rodada.
227     def process_sample(self):
228         s_wait_1 = 0; s_s_wait_1 = 0
229         s_wait_2 = 0; s_s_wait_2 = 0
230         s_server_1 = 0; s_server_2 = 0
231
232         # Loop que faz a soma e a soma dos quadrados
                dos tempos de espera e a soma dos tempos em
                servidor
233         # Dos clientes na fila 1 e na fila 2.
234         for client in self.clients:
235             s_wait_1 += client.wait(1)
236             s_s_wait_1 += client.wait(1)**2
237             s_server_1 += client.server[1]
238             s_wait_2 += client.wait(2)
239             s_s_wait_2 += client.wait(2)**2
240             s_server_2 += client.server[2]
241
242         # Adiciona a soma e a soma dos quadrados dos
                estimadores os valores estimados na rodada.
243         self.sums['m_s_W1'] += est.mean(s_wait_1, len(
                self.clients))
244         self.sums['m_s_s_W1'] += est.mean(s_wait_1, len
                (self.clients))**2
245         self.sums['v_s_W1'] += est.variance(s_wait_1,
                s_s_wait_1, len(self.clients))
246         self.sums['v_s_s_W1'] += est.variance(s_wait_1,
                 s_s_wait_1, len(self.clients))**2
247         self.sums['m_s_N1'] += est.mean(self.N_samples[
                'N_1'], self.t)
```

```
248         self.sums['m_s_s_N1'] += est.mean(self.
                N_samples['N_1'], self.t)**2
249         self.sums['m_s_Nq1'] += est.mean(self.N_samples
                ['Nq_1'], self.t)
250         self.sums['m_s_s_Nq1'] += est.mean(self.
                N_samples['Nq_1'], self.t)**2
251         self.sums['m_s_T1'] += est.mean(s_wait_1, len(
                self.clients)) + est.mean(s_server_1, len(
                self.clients))
252         self.sums['m_s_s_T1'] += (est.mean(s_wait_1,
                len(self.clients)) + est.mean(s_server_1,
                len(self.clients)))**2
253         self.sums['m_s_W2'] += est.mean(s_wait_2, len(
                self.clients))
254         self.sums['m_s_s_W2'] += est.mean(s_wait_2, len
                (self.clients))**2
255         self.sums['v_s_W2'] += est.variance(s_wait_2,
                s_s_wait_2, len(self.clients))
256         self.sums['v_s_s_W2'] += est.variance(s_wait_2,
                 s_s_wait_2, len(self.clients))**2
257         self.sums['m_s_N2'] += est.mean(self.N_samples[
                'N_2'], self.t)
258         self.sums['m_s_s_N2'] += est.mean(self.
                N_samples['N_2'], self.t)**2
259         self.sums['m_s_Nq2'] += est.mean(self.N_samples
                ['Nq_2'], self.t)
260         self.sums['m_s_s_Nq2'] += est.mean(self.
                N_samples['Nq_2'], self.t)**2
261         self.sums['m_s_T2'] += est.mean(s_wait_2, len(
                self.clients)) + est.mean(s_server_2, len(
                self.clients))
262         self.sums['m_s_s_T2'] += (est.mean(s_wait_2,
                len(self.clients)) + est.mean(s_server_2,
                len(self.clients)))**2
263         # Inicializa as estruturas de dados para a
                proxima rodada.
264         self.init_sample()
265
266     # Metodo que calcula os resultados (valor e
            intervalo de confianca) para cada estimador.
267     def calc_results(self):
```

```
268         self.results = {
269             'E[N1]'  : { 'value' : est.mean(self.sums['
                m_s_N1'], self.samples), 'c_i' : est.
                confidence_interval(self.sums['m_s_N1'],
                 self.sums['m_s_s_N1'], self.samples) },
270             'E[N2]'  : { 'value' : est.mean(self.sums['
                m_s_N2'], self.samples), 'c_i' : est.
                confidence_interval(self.sums['m_s_N2'],
                 self.sums['m_s_s_N2'], self.samples) },
271             'E[T1]'  : { 'value' : est.mean(self.sums['
                m_s_T1'], self.samples), 'c_i' : est.
                confidence_interval(self.sums['m_s_T1'],
                 self.sums['m_s_s_T1'], self.samples) },
272             'E[T2]'  : { 'value' : est.mean(self.sums['
                m_s_T2'], self.samples), 'c_i' : est.
                confidence_interval(self.sums['m_s_T2'],
                 self.sums['m_s_s_T2'], self.samples) },
273             'E[Nq1]' : { 'value' : est.mean(self.sums['
                m_s_Nq1'], self.samples), 'c_i' : est.
                confidence_interval(self.sums['m_s_Nq1'
                ], self.sums['m_s_s_Nq1'], self.samples)
                 },
274             'E[Nq2]' : { 'value' : est.mean(self.sums['
                m_s_Nq2'], self.samples), 'c_i' : est.
                confidence_interval(self.sums['m_s_Nq2'
                ], self.sums['m_s_s_Nq2'], self.samples)
                 },
275             'E[W1]'  : { 'value' : est.mean(self.sums['
                m_s_W1'], self.samples), 'c_i' : est.
                confidence_interval(self.sums['m_s_W1'],
                 self.sums['m_s_s_W1'], self.samples) },
276             'E[W2]'  : { 'value' : est.mean(self.sums['
                m_s_W2'], self.samples), 'c_i' : est.
                confidence_interval(self.sums['m_s_W2'],
                 self.sums['m_s_s_W2'], self.samples) },
277             'V(W1)'  : { 'value' : est.mean(self.sums['
                v_s_W1'], self.samples), 'c_i' : est.
                confidence_interval(self.sums['v_s_W1'],
                 self.sums['v_s_s_W1'], self.samples) },
278             'V(W2)'  : { 'value' : est.mean(self.sums['
                v_s_W2'], self.samples), 'c_i' : est.
```

```
                confidence_interval(self.sums['v_s_W2'],
                 self.sums['v_s_s_W2'], self.samples) }
279         }
280
281     # Metodo que atualiza a barra de progresso com o
            valor do menor intervalo de confianca encontrado
            a cada rodada.
282     def pb_amount(self):
283         return 1 - max((2.0*self.results['E[N1]']['c_i'
                ]/self.results['E[N1]']['value']), \
284                        (2.0*self.results['E[N2]']['c_i'
                           ]/self.results['E[N2]']['
                           value']), \
285                        (2.0*self.results['E[T1]']['c_i'
                           ]/self.results['E[T1]']['
                           value']), \
286                        (2.0*self.results['E[T2]']['c_i'
                           ]/self.results['E[T2]']['
                           value']), \
287                        (2.0*self.results['E[Nq1]']['c_i
                           ']/self.results['E[Nq1]']['
                           value']), \
288                        (2.0*self.results['E[Nq2]']['c_i
                           ']/self.results['E[Nq2]']['
                           value']), \
289                        (2.0*self.results['E[W1]']['c_i'
                           ]/self.results['E[W1]']['
                           value']), \
290                        (2.0*self.results['E[W2]']['c_i'
                           ]/self.results['E[W2]']['
                           value']), \
291                        (2.0*self.results['V(W1)']['c_i'
                           ]/self.results['V(W1)']['
                           value']), \
292                        (2.0*self.results['V(W2)']['c_i'
                           ]/self.results['V(W2)']['
                           value']))
293
294     # Metodo que testa se todos os intervalos de
            confianca sao validos.
295     # So faz a validacao a partir da terceira rodada.
```

```
296     def valid_confidence_interval(self):
297         return not(self.samples <= 2) and \
298                 (2.0*self.results['E[N1]']['c_i'] <=
                    0.1*self.results['E[N1]']['value'])
                    and \
299                 (2.0*self.results['E[N2]']['c_i'] <=
                    0.1*self.results['E[N2]']['value'])
                    and \
300                 (2.0*self.results['E[T1]']['c_i'] <=
                    0.1*self.results['E[T1]']['value'])
                    and \
301                 (2.0*self.results['E[T2]']['c_i'] <=
                    0.1*self.results['E[T2]']['value'])
                    and \
302                 (2.0*self.results['E[Nq1]']['c_i'] <=
                    0.1*self.results['E[Nq1]']['value'])
                    and \
303                 (2.0*self.results['E[Nq2]']['c_i'] <=
                    0.1*self.results['E[Nq2]']['value'])
                    and \
304                 (2.0*self.results['E[W1]']['c_i'] <=
                    0.1*self.results['E[W1]']['value'])
                    and \
305                 (2.0*self.results['E[W2]']['c_i'] <=
                    0.1*self.results['E[W2]']['value'])
                    and \
306                 (2.0*self.results['V(W1)']['c_i'] <=
                    0.1*self.results['V(W1)']['value'])
                    and \
307                 (2.0*self.results['V(W2)']['c_i'] <=
                    0.1*self.results['V(W2)']['value'])
308
309     # Metodo que exibe os resultados junto com o numero
            de rodadas processadas e os retorna.
310     def report(self):
311         print "Exibindo os resultados:"
312         for key in self.results.keys():
313             print key, ': ', self.results[key]['value'
                    ], ' - I.C: ', self.results[key]['c_i']
314         print "Numero de rodadas :", self.samples
315         return self.results
```

```
316
317     # Metodo que inicializa a lista dos clientes que
            serao testados.
318     def init_test(self):
319         for i in range(10):
320             self.test_list.append(math.floor(random.
                    random()*self.total_clients))
321
322     # Metodos que tratam o transito dos clientes das
            filas para o servidor, de acordo com a politica
            de atendimento usada.
323     @staticmethod
324     def pop_queue1_fcfs(instance):
325         return instance.queue1.popleft()
326
327     @staticmethod
328     def pop_queue2_fcfs(instance):
329         return instance.queue2.popleft()
330
331     @staticmethod
332     def pop_queue1_lcfs(instance):
333         return instance.queue1.pop()
334
335     @staticmethod
336     def pop_queue2_lcfs(instance):
337         return instance.queue2.pop()
```

7.1.2 Client

Classe que representa um cliente que entra no sistema. Possui seus tempos de entrada e saída da fila, tempo no servidor e cor.

```
1 class Client:
2     def __init__(self, color):
3         # Identificador do cliente, usada para o teste
              de correcao.
4         self.id = id
5         # Tempo de chegada ao servidor (fila 1 e fila
              2)
6         self.arrival = {}
```

```
7           # Tempo de saida do servidor (fila 1 e fila 2)
8           self.leave = {}
9           # Tempo no servidor (fila 1 e fila 2)
10          self.server = {}
11          # Indicador que diz qual fila o cliente esta no
                momento
12          self.queue = 0
13          # Indicador que diz se o cliente ja foi servido
                e saiu do sistema
14          self.served = 0
15          # Cor do cliente (TRANSIENT e EQUILIBRIUM)
16          self.color = color
17
18      def set_arrival(self, arrival):
19          self.arrival[self.queue] = arrival
20
21      def set_leave(self, leave):
22          self.leave[self.queue] = leave
23
24      def set_server(self, server):
25          self.server[self.queue] = server
26
27      def set_queue(self, queue):
28          self.queue = queue
29
30      def set_served(self, served):
31          self.served = served
32
33      # Tempo de espera na fila = Tempo de saida da fila
            para o servidor - Tempo de chegada na fila.
34      def wait(self, queue):
35          return (self.leave[queue] - self.arrival[queue
                ])
```

7.1.3 EventHeap

Classe que representa a lista de eventos que é processada durante uma rodada de simulação.

```
1 import heapq
```

```
2
3
4 class EventHeap(list):
5     # Adicionar evento a lista
6     def push(self, (time, event_type)):
7         heapq.heappush(self, (time, event_type))
8
9     # Remover evento da lista
10    def pop(self):
11        return heapq.heappop(self)
```

7.1.4 Analytic

Classe que serve para calcular os resultados de forma analítica.

```
1 from util.constants import *
2
3
4 class Analytic:
5
6     def __init__(self, entry_rate, service_policy,
          service_rate=1.0):
7         self.entry_rate = entry_rate
8         self.service_policy = service_policy
9         self.service_rate = service_rate
10        self.utilization = 2.0*(entry_rate/service_rate
              )
11        self.X = 1.0/self.service_rate
12        self.results = { 'E[W1]' : 0.0, 'E[W2]' :
              0.0, 'E[T1]' : 0.0, 'E[T2]' : 0.0,
13                         'E[Nq1]' : 0.0, 'E[Nq2]' :
                               0.0, 'E[N1]' : 0.0, 'E[N2]
                               ' : 0.0,
14                         'V(W1)' : 0.0, 'V(W2)' : 'X'
                               }
15
16    # Metodo que define os valores de forma analitica.
17    def start(self):
18        self.results['E[W1]'] = (self.utilization*self
              .X)/(1.0 - self.entry_rate*self.X)
```

```
19          self.results['E[W2]']  = (self.utilization*self
               .results['E[W1]'] + 2.0*self.entry_rate*(
               self.service_rate**2))/(1.0 - self.
               utilization)
20          self.results['E[T1]']  = self.results['E[W1]']
               + self.X
21          self.results['E[T2]']  = self.results['E[W2]']
               + self.X
22          self.results['E[Nq1]'] = self.entry_rate*self.
               results['E[W1]']
23          self.results['E[Nq2]'] = self.entry_rate*self.
               results['E[W2]']
24          self.results['E[N1]']  = self.entry_rate*self.
               results['E[T1]']
25          self.results['E[N2]']  = self.entry_rate*self.
               results['E[T2]']
26          if self.service_policy == FCFS:
27              self.results['V(W1)'] = (4.0*self.
                   utilization)/(2.0 - self.utilization)
28          elif self.service_policy == LCFS:
29              self.results['V(W1)'] = (4.0*self.
                   entry_rate)*(self.entry_rate**2 - self.
                   entry_rate + 1)/((1.0 - self.entry_rate)
                   **3)
30
31      # Metodo que exibe os resultados encontrados e os
           retorna.
32      def report(self):
33          print "Exibindo os resultados analiticos: "
34          for key in self.results.keys():
35              print key, ': ', self.results[key]
36
37          return self.results;
```

7.1.5 ResultParser

Classe que formata os resultados encontrados em um documento .html usando o parser DOM.

```
1 from xml.dom.minidom import *
```

```
2 from util.constants import *
3
4
5 class ResultParser:
6
7     def __init__(self, results):
8         self.results = results
9         self.doc = Document()
10
11     # Metodo que cria a estrutura html do documento e
          retorna o elemento <body>
12     def create_header(self):
13         html = self.doc.createElement('html')
14         header = self.doc.createElement('header')
15         title = self.doc.createElement('title')
16         title_text = self.doc.createTextNode("Tabelas␣
              com␣os␣resultados␣da␣simulacao")
17         body = self.doc.createElement('body')
18         self.doc.appendChild(html)
19         html.appendChild(header)
20         html.appendChild(body)
21         header.appendChild(title)
22         title.appendChild(title_text)
23
24         return body
25
26     # Metodo que cria as tabelas
27     def create_table(self, table_type, name):
28         table = self.doc.createElement('table')
29         table.setAttribute('cellspacing', '0')
30         table.setAttribute('cellpadding', '4')
31         table.setAttribute('border', '1')
32
33         tr = self.doc.createElement('tr')
34         th = self.doc.createElement('th')
35         th.setAttribute('align', 'center')
36         th.setAttribute('colspan', '31')
37         th.appendChild(self.doc.createTextNode("Tabela␣
              com␣os␣resultados␣para␣a␣politica␣de␣
              atendimento␣" + name))
38         tr.appendChild(th)
```

```
39          table.appendChild(tr)
40
41          tr = self.doc.createElement('tr')
42          th = self.doc.createElement('th')
43          th.setAttribute('align', 'center')
44          th.setAttribute('style', 'font-weight: bold')
45          th.appendChild(self.doc.createTextNode("uti."))
46          tr.appendChild(th)
47          headers = ['E[N1]', 'E[N2]', 'E[T1]', 'E[T2]',
                'E[Nq1]', 'E[Nq2]', 'E[W1]', 'E[W2]', 'V(W1)
                ', 'V(W2)']
48          for header in headers:
49              th = self.doc.createElement('th')
50              th.setAttribute('align', 'center')
51              th.appendChild(self.doc.createTextNode(
                    header))
52              tr.appendChild(th)
53          table.appendChild(tr)
54
55          utilizations = self.results[table_type].keys()
56          utilizations.sort()
57          for utilization in utilizations:
58              tr = self.doc.createElement('tr')
59              td = self.doc.createElement('td')
60              td.setAttribute('align', 'center')
61              td.setAttribute('style', 'font-weight: bold
                    ')
62              td.appendChild(self.doc.createTextNode(str(
                    utilization)))
63              tr.appendChild(td)
64              for key in headers:
65                  td = self.doc.createElement('td')
66                  td.setAttribute('align', 'center')
67                  div = self.doc.createElement('div')
68                  div.setAttribute('style', 'padding:5px;
                        ')
69                  if type(self.results[table_type][
                        utilization]['analytic'][key]).
                        __name__ == 'str':
70                      div.appendChild(self.doc.
                            createTextNode(str(self.results[
```

```
                        table_type][utilization]['
                        analytic'][key])))
71                  else:
72                      div.appendChild(self.doc.
                            createTextNode(str(round(float(
                            self.results[table_type][
                            utilization]['analytic'][key]),
                            5))))
73                  td.appendChild(div)
74                  div = self.doc.createElement('div')
75                  div.setAttribute('style', 'padding:5px;
                        border-top: 1px solid #000000;
                        border-bottom: 1px solid #000000')
76                  div.appendChild(self.doc.createTextNode
                        (str(round(float(self.results[
                        table_type][utilization]['simulator'
                        ][key]['value']), 5))))
77                  td.appendChild(div)
78                  div = self.doc.createElement('div')
79                  div.setAttribute('style', 'padding:5px;
                        ')
80                  div.appendChild(self.doc.createTextNode
                        (str(round(float((2.0*self.results[
                        table_type][utilization]['simulator'
                        ][key]['c_i']/self.results[
                        table_type][utilization]['simulator'
                        ][key]['value'])*100.0), 2)) + "%"))
81                  td.appendChild(div)
82                  tr.appendChild(td)
83              table.appendChild(tr)
84
85          return table
86
87      # Metodo que cria o documento html usando os
            resultados dados
88      def parse(self):
89          body = self.create_header()
90          table_fcfs = self.create_table(FCFS, 'F.C.F.S')
91          table_lcfs = self.create_table(LCFS, 'L.C.F.S')
92          table_lcfs.setAttribute('style', 'margin-top
                :100px')
```

```
93          body.appendChild(table_fcfs)
94          body.appendChild(table_lcfs)
95
96      # Metodo que escreve o documento DOM gerado em um
            arquivo .html no disco
97      def write(self, filename):
98          file = open(filename, "w")
99          print >>file, self.doc.toprettyxml()
100         file.close()
```

7.2 Modulos utilitários

7.2.1 Estimator

Módulo que possui métodos para retornar os estimadores de média, variância e calcula intervalos de confiança.

```
1 import math
2 import scipy.stats
3
4
5 # Retorna o valor t de student para um intervalo de
      confianca de 95% e [samples] amostras.
6 def t_st_value(samples):
7     return scipy.stats.t.ppf(0.975, samples)
8
9 # Retorna a media estimada usando a soma [sum] dos
      valores calculados e o numero total [samples] de
      valores.
10 def mean(sum, samples):
11     return sum/float(samples)
12
13 # Retorna a variancia estimada usando a forma
      incremental usando a soma [sum] dos valores, a soma
      dos quadrados [square_sum]
14 # e o numero total [samples] de valores.
15 def variance(sum, square_sum, samples):
16     return square_sum/float(samples-1) - (sum**2)/float
          (samples*(samples-1))
17
```

```
18 # Retorna o limite do intervalo de confianca  usando a
       soma [sum] dos valores e a soma dos quadrados [
       square_sum]
19 # para calcular o desvio padrao e o numero de rodadas [
       samples].
20 def confidence_interval(sum, square_sum, samples):
21     std_deviation = math.sqrt(variance(sum, square_sum,
           samples))
22     return (t_st_value(samples)*std_deviation)/math.
          sqrt(samples)
23
24 if __name__ == "__main__":
25     print "Testando..."
26     list1 = [11.0, 5.0, 10.0, 9.0, 15.0, 6.0, 18.0,
          8.0, 12.0, 9.0, 5.0, 10.0, 7.0, 13.0, 15.0]
27     list2 = [10.0, 2.0, 15.0, 4.0, 5.0, 16.0, 8.0, 4.0,
           2.0, 19.0, 10.0, 2.0, 9.0, 10.0, 12.0]
28     print "Mean sample 1: ", mean(sum(list1), len(list1
          ))
29     print "Mean sample 2: ", mean(sum(list2), len(list2
          ))
30     print "Samples mean: ", mean((mean(sum(list1), len(
          list1)) + mean(sum(list2), len(list2))), 2)
31     print "Samples variance: ", variance((mean(sum(
          list1), len(list1)) + mean(sum(list2), len(list2
          ))), ((mean(sum(list1), len(list1))**2) + (mean(
          sum(list2), len(list2))**2)), 2)
32     print "Student's T value: ", t_st_value(10000)
```

7.2.2 Distribution

Módulo com o método que retorna os tempos aleatórios de chegada de uma distribuição exponencial.

```
1 import math
2 import random
3 import estimator
4
5
```

```
6 # Retorna um tempo aleatorio de uma distribuicao
      exponencial com taxa [rate].
7 def exp_time(rate):
8         return -(math.log(1.0 - random.random())/rate)
9
10 if __name__ == "__main__":
11         print "Testando..."
12         print estimator.mean(exp_time(2, 650))
```

7.2.3 Constants

Módulo que declara as constantes que são utilizadas pelo simulador.

```
1 # Constantes utilizadas para os tipos de eventos
      tratados pelo simulador
2 INCOMING = 1
3 SERVER_OUT = 2
4 SERVER_1_IN = 3
5 SERVER_2_IN = 4
6
7 # Constantes utilizadas para definir as cores dos
      clientes
8 TRANSIENT = 1
9 EQUILIBRIUM = 2
10
11 # Constantes utilizadas para definir a politica de
      atendimento das filas
12 FCFS = 1
13 LCFS = 2
```

7.2.4 Plot

Módulo que usa a biblioteca matplotlib para desenhar os gráficos necessários para a estimativa da fase transiente (Não é utilizado na versão final).

```
1 from math import sin, cos
2 import matplotlib.pyplot as plt
3
4
```

```
5 def plot(list, *args, **kwargs):
6     plt.plot(xrange(len(list)), list, *args, **kwargs)
7
8 def show(title):
9     plt.title(title)
10    plt.grid()
11    plt.show()
12
13 if __name__ == "__main__":
14    x = range(100)
15    y = [sin(item) for item in range(100)]
16    z = [cos(item) for item in range(100)]
17    plot(x, y, 'b-')
18    plot(x, z, 'r-')
19    show()
```

7.2.5 ProgressBar

Biblioteca usada para a construção da barra de progresso usada para efeito de visualização do progresso do processamento das rodadas do simulador.

Seu código não é apresentado aqui porque ele não foi escrito por nós e os seus créditos são devidamente citados em comentários no próprio fonte.

7.3 Principal

Este é o código que executa o simulador e faz os cálculos analíticos para todo o tipo de experimento requisitado, é dado como entrada o número de clientes que serão avaliados a cada rodada. Os resultados finais gerados pelo simulador e os resultados dos cálculos analíticos são gravados em tabelas no formato .html ao final da execução.

```
1 import time, sys, os, psyco
2 from obj.simulator import *
3 from obj.analytic import *
4 from obj.result_parser import *
5
6 # O psyco e um modulo que agiliza a execucao do codigo.
7 psyco.full()
8
```

```
9  if __name__ == "__main__":
10     clients = input("Entre com o numero de clientes que
           serao avaliados:")
11     #dados de entrada (taxa de entrada e valor da fase
           transiente)
12     entry_data = [[0.1, 30000], [0.2, 40000], [0.3,
           80000], [0.4, 400000], [0.45, 500000]]
13     service_policies = [
14         { 'value' : FCFS, 'name' : "F.C.F.S (First Come
               First Served)" },
15         { 'value' : LCFS, 'name' : "L.C.F.S (Last Come
               First Served)"  }
16     ]
17
18     # Dicionario que ira guardar os resultados
           adquiridos pelo simulador e pelo calculo
           analitico
19     results = {
20         FCFS : {
21             0.2 : { 'simulator' : {}, 'analytic' : {}
                   },
22             0.4 : { 'simulator' : {}, 'analytic' : {}
                   },
23             0.6 : { 'simulator' : {}, 'analytic' : {}
                   },
24             0.8 : { 'simulator' : {}, 'analytic' : {}
                   },
25             0.9 : { 'simulator' : {}, 'analytic' : {} }
26         },
27         LCFS : {
28             0.2 : { 'simulator' : {}, 'analytic' : {}
                   },
29             0.4 : { 'simulator' : {}, 'analytic' : {}
                   },
30             0.6 : { 'simulator' : {}, 'analytic' : {}
                   },
31             0.8 : { 'simulator' : {}, 'analytic' : {}
                   },
32             0.9 : { 'simulator' : {}, 'analytic' : {} }
33         }
34     }
```

```
35
36      # Loop que ira rodar o simulador para todos os
            casos requeridos
37      for entry_datum in entry_data:
38          entry_rate = entry_datum[0]
39          warm_up = entry_datum[1]
40          for service_policy in service_policies:
41              print "taxa de entrada =", entry_rate
42              print "tamanho da fase transiente =",
                    warm_up
43              print "politica de atendimento =",
                    service_policy['name']
44
45              print "Iniciando simulacao:"
46              # Chamada e execucao do simulador
47              simulator = Simulator(entry_rate=entry_rate
                    , warm_up=warm_up, clients=clients,
                    service_policy=service_policy['value'])
48              os.system("date")
49              start = time.time()
50              # Bind ao psyco para acelerar a execucao da
                     logica do simulador
51              psyco.bind(simulator.start)
52              simulator.start()
53              finish = time.time()
54              print "Tempo total de execucao :", (finish
                    - start)
55              results[service_policy['value']][2.0*
                    entry_rate]['simulator'] = simulator.
                    report()
56
57              print "Iniciando calculo analitico:"
58              # Chamada e execucao da classe que executa
                    os calculos analiticos
59              analytic = Analytic(entry_rate=entry_rate,
                    service_policy=service_policy['value'])
60              analytic.start()
61              results[service_policy['value']][2.0*
                    entry_rate]['analytic'] = analytic.
                    report()
62
```

```
63      print "Gerando as tabelas com os resultados"
64      # Chamada e execucao da classe que formata os
            resultados encontrados em um arquivo html
65      parsed_result = ResultParser(results)
66      parsed_result.parse()
67      parsed_result.write('resultados.html')
68      print "Tabelas geradas com sucesso no arquivo '
            resultados.html'."
```

7.4 Testes

Rotina que executa os testes de correção explicados na seção .

```
1 from obj.simulator import *
2
3
4 if __name__ == "__main__":
5     print "Testando a correcao do simulador..."
6     entry_data = [[0.1, 30000], [0.2, 40000]]
7     service_policies = [
8         { 'value' : FCFS, 'name' : "F.C.F.S (First Come
              First Served)" },
9         { 'value' : LCFS, 'name' : "L.C.F.S (Last Come
              First Served)"  }
10    ]
11    clients = 100000
12
13    for entry_datum in entry_data:
14        entry_rate = entry_datum[0]
15        warm_up = entry_datum[1]
16        for service_policy in service_policies:
17            print "teste com taxa", entry_rate, ",
                  tamanho de fase transiente igual a",
                  warm_up, ",", clients, "clientes
                  avaliados", \
18                    "e politica de atendimento",
                          service_policy['name']
19            simulator = Simulator(entry_rate=entry_rate
                  , warm_up=warm_up, clients=clients,
                  service_policy=service_policy['value'],
```

```
                test=True)
20              simulator.start()
21              simulator.report()
```