Modelos de Simulação por Eventos Discretos Baseados na Linguagem Livre Python

Aluno: Kevin Pires Novaes. R.A.: 177725.
Orientador: Prof. Dr. Edson Luiz Ursini.

1. Introdução

O objetivo deste projeto é avaliar por meio de testes utilizando conceitos de simulação por eventos discretos, a capacidade da linguagem Python em desempenhar tarefas facilmente implementadas utilizando o software Arena, para com isso amplificar as possibilidades de desenvolvimento de códigos com esse propósito, visto que a linguagem Python é livre e de código aberto, possibilitando maior integração entre programadores de diferentes realidades, tendo em vista que, o software Arena é privado e nem todos os estudantes e atuantes nesta área de desenvolvimento podem comprar a licença necessária.

2. Materiais e Métodos

A elaboração deste projeto de pesquisa consistiu em 4 etapas: análise de bibliotecas e *frameworks* específicos da linguagem Python no contexto da simulação por eventos discretos, depois foi feita a verificação de qual experimento era mais viável para simulação tanto no Python como no Arena, seguido da simulação de fato em ambos os ambientes, e por fim, a comparação dos resultados/desempenho entre as simulações feitas com o Arena e com a linguagem Python.

2.1. Verificação do experimento mais viável

Por meio de uma série de testes, verificou-se que a melhor alternativa visando simplicidade e efetividade nos resultados objetivados, seria a simulação do sistema de filas M/M/1. Todo programa de simulação para poder ser aplicado precisa ser validado. Como o modelo M/M/1 é conhecido e tem solução matemática fechada, o modelo de simulação pode ser comparado com o resultado analítico para que seja validado. Uma vez validado, é relativamente fácil incrementá-lo para contemplar outras distribuições de chegada e/ou de serviço que não têm solução explícita, ou que só podem ser calculados numericamente (como exemplo, os modelos M/G/1, com distribuição geral para o serviço, e o G/G/1, com distribuição geral para as chegadas e para o serviço). Na verdade, nos casos reais, a melhor distribuição deve ser obtida por medidas efetuadas no sistema real.

O sistema de filas M/M/1 é caracterizado pela distribuição exponencial de chegadas e tempos de serviço, considerando por exemplo, uma fila de transmissão única alimentando um único *link*, ou seja, um servidor. É importante definir o caráter exponencial das chegadas, pois assume um processo de Poisson, e serviço exponencial, conhecido por tratar e modelar situações reais de maneira estocástica com um ótimo nível de aproximação, caso algumas condições sejam cumpridas pelo sistema. Essas condições para o contexto de filas, são:

1. O número de clientes no sistema deve ser muito grande (população infinita);
2. O impacto de um único cliente no desempenho do sistema deve ser muito pequeno, ou seja, um único cliente deve consumir uma porcentagem muito baixa dos recursos do sistema como um todo;
3. Todos os clientes são independentes, ou seja, a decisão de um cliente usar o sistema independe da decisão de outros.

Alguns sistemas que cumprem com essas condições são: chegada de ligações telefônicas em uma central de telefonia, chegada de clientes em uma rede de supermercados considerando um dia normal, duração de chamadas telefônicas, atraso na transmissão de dados, etc.

2.2. Simulação da fila M/M/1 utilizando Python e Arena

O código específico utilizado como referência para a simulação com a linguagem Python foi feito em Python 3.5 utilizando a plataforma de desenvolvimento Pycharm pelo programador Robert BASOMINGERA, como um projeto na área de 'Radio Resources Management' para a Ajou University (<https://github.com/basomingera/MM1-queue-simulator>). Por outro lado, a simulação feita com o software Arena, foi realizada via interface gráfica, utilizando um diagrama de blocos.

2.3. Comparação dos resultados/desempenho

Para que se validem os resultados do modelo de fila M/M/1 é preciso conhecer alguns conceitos conhecidos (fórmulas), tais como:

- Intensidade de tráfego (ou ocupação): definida como a taxa média de chegadas (lambda) dividida pela taxa média de serviço (μ).
$\rho = \frac{\lambda}{\mu}$ (3);
- Número médio de clientes no sistema (N): $N = \frac{\rho}{1-\rho}$ (4);
- Tempo de serviço (Ts): $Ts = \frac{1}{\mu}$ (5);
- Tempo total de espera (incluindo tempo de serviço): $T = \frac{1}{\mu-\lambda}$ (6).

Importante perceber que a taxa média de serviço (μ) sempre deve ser maior que a taxa média de chegadas (λ), de maneira que a intensidade de tráfego seja sempre menor que 1. Esse fundamento é importante e significa a estabilidade do sistema de filas, pois à medida que a ocupação (rho=λ/μ) se aproxima de 1, o número médio de clientes (N) tende ao infinito, fazendo com que o tempo total de espera aumente proporcionalmente.

3. Resultados e discussão

Nesta seção serão apresentados os resultados referentes às simulações feitas tanto no Arena como na linguagem Python, relacionando os resultados teóricos calculados por meio das fórmulas da fila M/M/1 explicadas anteriormente com os

resultados simulados. Para a simulação, foram utilizados os valores de μ (1/10 = 0.1) e λ (0.03, 0.05, 0.07, 0.09) para verificar se o tempo total de espera simulado é igual ou próximo ao teórico.

3.1. Resultados teóricos

O único valor constante em toda a simulação foi o tempo de serviço, $Ts = 10$ segundos (μ=1/10, μ=0.1). Os valores de lambda foram variando de acordo com a tabela abaixo que relaciona os mesmos com a taxa média de serviço, tempo total de espera em segundos (T(s)) e número médio de clientes no sistema (N), estes calculados utilizando as equações (4), (5) e (6) conforme a Tabela 1:

Tabela 1 - Valores teóricos.

λ	0,03	0,05	0,07	0,09
μ	0,10	0,10	0,10	0,10
ρ	0,3	0,5	0,7	0,9
N	0,43	1	2,33	9
$T(s)$	14,29	20,00	33,33	99,99

3.2. Resultados Python

O código escrito em Python citado anteriormente foi executado utilizando Python 3.8.5 através do sistema operacional Arch Linux. Segue o resultado do console:

Time: 0 sec, Simulation starts for **λ=0.03 (Fig. 10)**
Time: 100001.53987599126secs *End of last job in the System*
Simulation summary:
Total jobs: **3054**
Average delay per job: **14.039494325819888**

Time: 0 sec, *Simulation starts for* **λ=0.05 (Fig. 11)**
Time: 100000secs *End of last job in the System*
Simulation summary:
Total jobs: **5015**
Average delay per job: **20.099737978000288**

Time: 0 sec, *Simulation starts for* **λ=0.07 (Fig. 12)**
Time: 100004.5692551423secs *End of last job in the System*
Simulation summary:
Total jobs: **7009**
Average delay per job: **34.157419863799134**

Time: 0 sec, *Simulation starts for* **λ=0.09 (Fig. 13)**
Time: 100312.10531839152secs *End of last job in the System*
Simulation summary:
Total jobs: **9040**

Average delay per job: **97.83316677610286**

A Fig. 1 mostra o valor do tempo de espera médio simulado e teórico para cada um dos valores de λ.

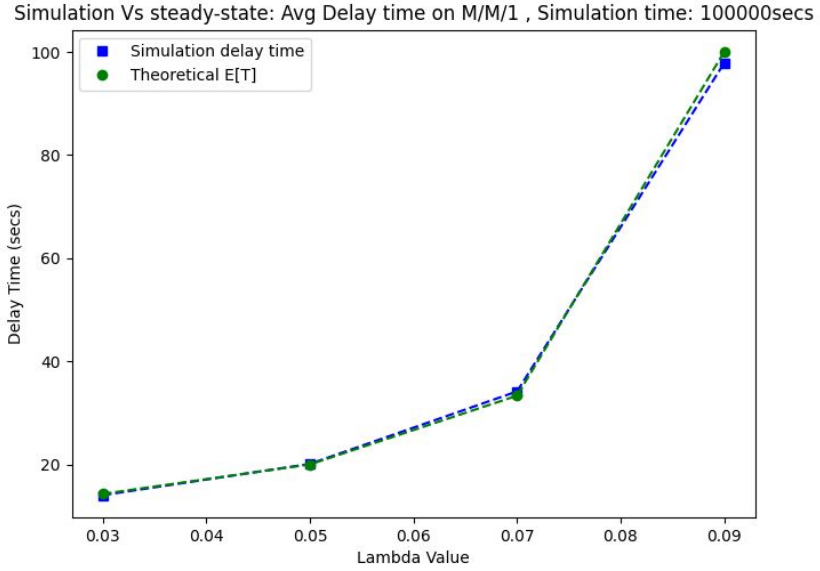
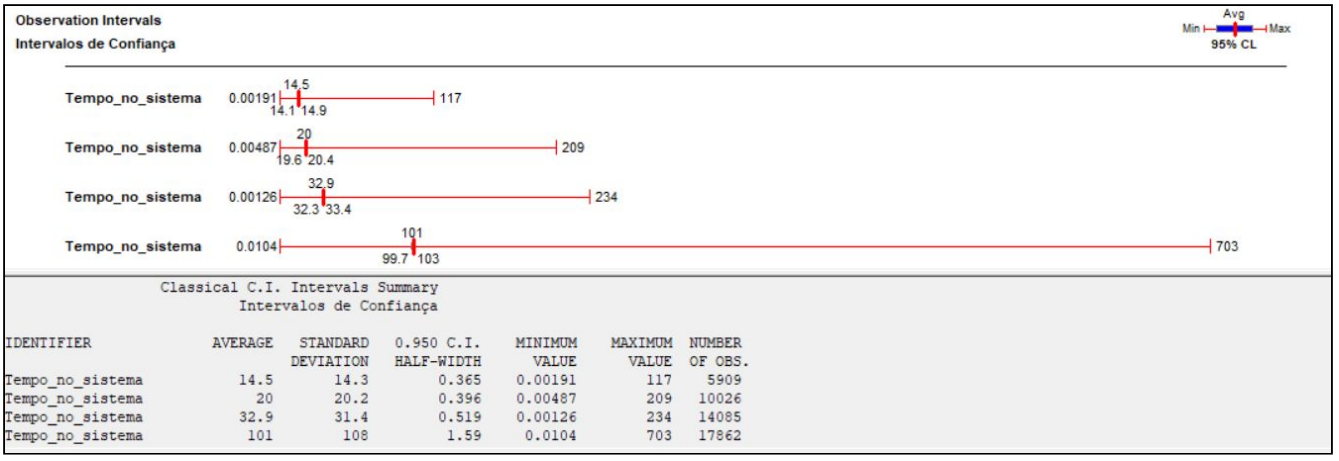


Figura 1 - Tempo de espera x Valores de lambda (0.03, 0.05, 0.07, 0.09 - Python).

3.3. Resultados Arena

Em resumo tem-se o gráfico na Fig. 2 que sintetiza as informações da simulação no Arena utilizando o dobro do tempo utilizado com o Python, logo o número de processos (Jobs) deve ser aproximadamente o dobro.



Classical C.I. Intervals Summary
Intervalos de Confiança

IDENTIFIER	AVERAGE	STANDARD DEVIATION	0.950 C.I. HALF-WIDTH	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBS.
Tempo_no_sistema	14.5	14.3	0.365	0.00191	117	5909
Tempo_no_sistema	20	20.2	0.396	0.00487	209	10026
Tempo_no_sistema	32.9	31.4	0.519	0.00126	234	14085
Tempo_no_sistema	101	108	1.59	0.0104	703	17862

Figura 2 - Valores finais da simulação com o Arena para 200000 segundos.

A Fig. 3 mostra a comparação entre os valores teóricos e simulados do tempo de espera médio para os valores de lambda.

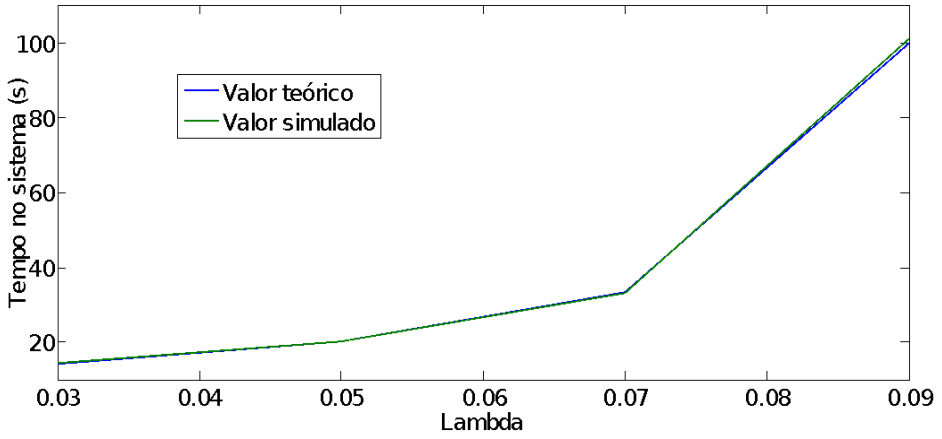


Figura 3 - Tempo de espera x Valores de λ (0.03, 0.05, 0.07, 0.09 - Arena).

3.4. Comparação dos resultados

Segue uma síntese dos tempos de espera teóricos e simulados (Tabela 2):

Tabela 2 - Tempos de espera total (Teórico, Arena e Python).

λ	0,03	0,05	0,07	0,09
T(s)	14,29	20,00	33,33	99,99
T(s), Arena	**14.5**	**20**	**32.9**	**101**
T(s), Python	**14.04**	**20.01**	**34.16**	**97.83**

4. Conclusões

Em suma, fica claro pelos resultados obtidos que tanto o Python quanto o Arena possuem um bom ambiente para simulação por eventos discretos, disponibilizando uma boa aproximação em comparação com valores pré calculados teoricamente. Importante frisar que neste projeto em questão, utilizamos um modelo teórico simples e altamente testado com fórmulas de fácil aplicação, de maneira que, a eficácia dos resultados não pode ser generalizada para qualquer modelo analítico.

O objetivo deste projeto era comprovar através de resultados analíticos se o uso de uma linguagem livre e de código aberto como o Python é interessante no contexto de simulação por eventos discretos, já que o software Arena é o mais usado para esse fim, e este é um software proprietário. Por fim, foi possível perceber que a linguagem Python é funcional e pode ser incorporada em projetos e/ou trabalhos que envolvam esse tema.