UFRJ

UNIVERSIDADE FEDERAL
DO RIO DE JANEIRO

Guilherme Vinicius de Oliveira Soares

# Modelagem e Avaliação de Desempenho

## Simulação de Sistema de Filas Aberto

Rio de Janeiro, 11 de dezembro de 2024

# 1 - Introdução

Ao longo do semestre aprendemos teoricamente sobre modelagem de filas, seus usos, e aplicações no mundo real. Ela auxilia, entre outras coisas: a identificar gargalos em um sistema, prever ou simular condições de carga para testar limites de um sistema, promover balanceamento de carga, etc…

A partir deste modelo teórico, existem situações onde uma modelagem estritamente teórica baseada em matemática e estatística se prova inviável de modelar em tempo hábil por questões de complexidade ou pelo sistema não ter um comportamento matematicamente bem definido. Para estes casos uma simulação prática por vezes proporciona resultados satisfatórios o suficiente com certa margem de erro em relação aos modelos teóricos fechados.

Com isso, a intenção deste trabalho prático é simular um sistema em particular.

# 2 - Descrição do Cenário

O objetivo do trabalho é modelar um sistema de filas como na figura:

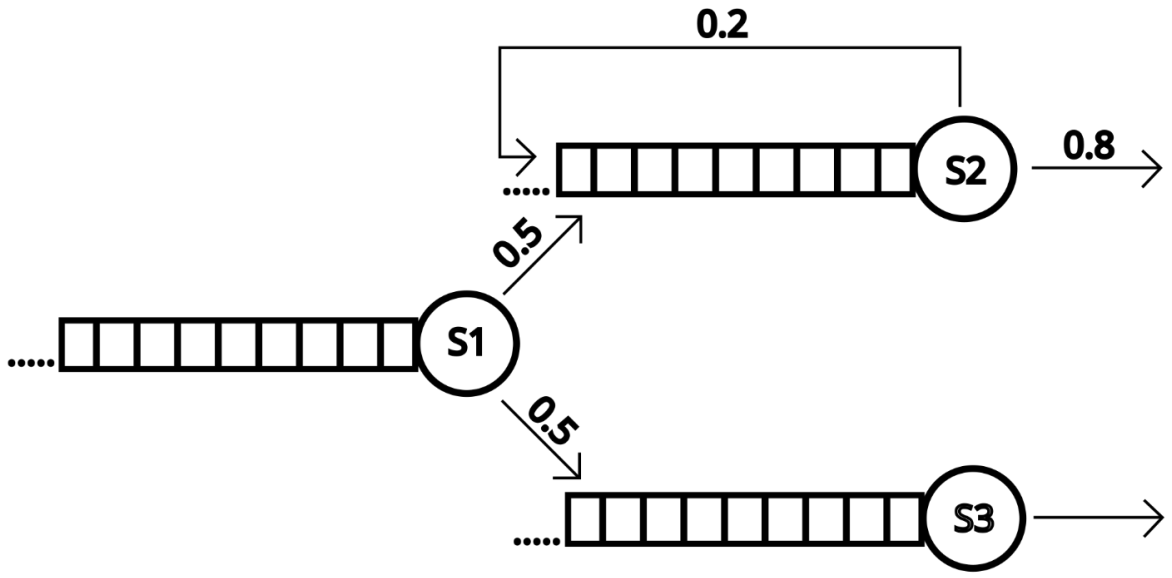


Figura 1 - Sistema de filas do enunciado

Trata-se de um sistema de filas de um sistema aberto com as propriedades: composto por 3 servidores: S1, S2 e S3, todos com fila de capacidade infinita. Quando um processo novo chega, ele é colocado na fila de S1, após seu processamento por S1 ele vai para a fila de S2 ou S3 com 50% de probabilidade cada. Um processamento em S2 resulta em um retorno a S2 com 20% de probabilidade, ou saída do sistema com 80% de probabilidade. Um processamento concluído em S3 sai do sistema deterministicamente.

Ainda neste cenário, faremos simulações de 3 formas diferentes, variando as funções que governam o tempo médio de serviço para cada um dos servidores, seguindo a ordem:

- Primeiro cenário: tempos de serviço determinísticos:
    - 0.4s para o Servidor 1
    - 0.6s para o Servidor 2
    - 0.95s para o Servidor 3

- Segundo cenário: tempos de serviço determinados por sorteio em distribuições uniformes
    - ***U***(0.1, 0.7) para o Servidor 1
    - ***U***(0.1, 1.1) para o Servidor 2
    - ***U***(0.1, 1.8) para o Servidor 3

- Terceiro cenário: tempos de serviço determinadas por Variáveis Aleatórias exponenciais com:
    - Média 0.4s para o Servidor 1
    - Média 0.6s para o Servidor 2
    - Média 0.95s para o Servidor 3

Além disso, para todos os cenários a taxa de chegada de processos é determinada também por uma Variável Aleatória exponencial com *lambda* = 2.

O objetivo principal é calcular a média e o desvio padrão do tempo total de serviço para cada situação.

# 3 - Implementação

A implementação foi feita na linguagem de programação Python utilizando em especial o paradigma de Orientação a Objetos. Também foram utilizadas as bibliotecas *Pandas* e *random*, a segunda sendo interna do Python sem necessidade de instalação.

## 3.1 - Processos

Buscando se aproximar ao máximo de um sistema real, primeiro foi modelado um objeto para representar o processo em si. Ele é inicializado com variáveis que guardam o tempo total decorrido em cada fila e cada servidor individualmente, sendo inicializados com 0. Na hora de sua criação também é passado um parâmetro que simboliza o tempo “real” discretizado em que o processo foi criado, servindo de auxílio para calcular o tempo em filas.

Foi também criado uma única função que é acionada quando o processo está prestes a sair do sistema, ela soma os tempos em filas, tempos em servidores e novamente os dois anteriores para produzir o tempo total no sistema. Essas métricas são então

inseridas num *Data Frame* que contém as métricas de todos os processos para um determinado cenário.

## 3.2 - Filas

Filas para os servidores foram implementadas em *arrays* comuns da linguagem Python, os métodos *Pop* e *Append*, ambos implementados *out-of-the-box* são tudo que precisamos para implementar a fila do tipo FIFO (First In First Out) do problema. É utilizado um *array* para cada servidor para fazer o papel de fila.

## 3.3 - Tempos de chegada

Lidar com tempos de chegada gerados por variáveis aleatórias possivelmente seria complicado num sistema que funciona em tempo discretizado, para remover um pouco da complexidade da modelagem utilizamos um método que facilita a implementação: ao invés de sortear as variáveis de chegada ao longo da execução da simulação, sorteamos os N tempos de chegada desejados e os armazenamos num *array* extra auxiliar. Façamos então um laço de forma que adicionamos à posição i+1 do *array* o valor contido na posição i, iterando do primeiro ao penúltimo.

Desta forma, cada tempo de chegada será a soma de si mesmo com todas as chegadas anteriores. Podemos implementar facilmente a chegada de processos com o nosso relógio interno discreto, sem a necessidade de criar relógios adicionais. Já que o *array* estará em ordem crescente, basta olhar se o tempo no nosso relógio é maior ou igual ao tempo contido na primeira posição do *array.* Se sim, adicionamos um processo novo à fila do primeiro servidor e descartamos (damos *pop*) a primeira posição do *array* de tempos de chegada. Esse valor que é descartado é utilizado na criação do objeto Processo, já que ele fornece o tempo em que o processo chega na primeira fila.

Como estados interessados em analisar N processos que *saíram* do sistema e não em interromper o sistema quando o processo de número N entrar nele, adicionamos também um número arbitrário de processos extras.

## 3.4 - Servidores

Foi criado um objeto diferente para cada servidor, mas abordaremos primeiro seus pontos comuns.

## 3.4.1 - Criação

No momento de sua criação, o único parâmetro passado para o construtor de um servidor é o “*mode*”, ele serve para *setar* qual das opções de tempo de serviço serão utilizadas ao longo do experimento. Também são criadas as variáveis internas: *processo,* para armazenar o processo que vier da fila, *tempo Requerido*, para armazenar o tempo total que um processo deve ficar no servidor e *tempo Decorrido,* para comparar com a variável anterior e decidir se um processo deve avançar a próxima etapa ou não.

## 3.4.2 - Funções

Foi criada a função *service Time* que é chamada toda vez que um servidor começa a trabalhar num novo processo, dependendo da opção de “*mode*” que foi passada no ato da criação do objeto, a variável *tempo Requerido* é setada para uma das formas de tempo de serviço determinadas no item 2, dependendo do servidor.

Cada servidor também possui a função *iterate* que é o coração de todo o experimento. A função é chamada para cada servidor com um valor de passagem de tempo discreto (optei por 0.01 segundos) que incrementará o tempo decorrido.
Caso um servidor não tenha processo corrente e a fila esteja vazia a função apenas retorna. Caso um servidor não tenha processo corrente e exista um processo na fila, o processo é movido para dentro do servidor, o tempo de fila é descartado (como estamos utilizando tempo discreto, um processo iniciado a partir de um ponto onde não há outro processo no servidor, significa na verdade que ele não passou tempo na fila), o *tempo Decorrido* é zerado, e um novo *tempo Requerido* é sorteado.

A discrepância entre cada servidor ocorre apenas quando a função *iterate* se depara com um caso onde um processo terminou, ou em outras palavras, quando o *tempo Decorrido* for maior ou igual que o *tempo Requerido*. Para os três servidores, o *tempo Requerido,* é somado para a variável de tempo decorrido interna do Processo, para o servidor respectivo. O tempo inicial de fila de um Processo também é alterado para o tempo atual.

Se o processo finalizar no Servidor 1, ele é colocado na fila do Servidor 2 ou do Servidor 3 com 50% de chance cada com o método *Append*. Se um processo finalizar no Servidor 2, com 20% de chance ele é recolocado na fila do Servidor 2, caso contrário o processo sai do sistema (e a função interna de resultados do Processo é chamada). Por fim, se o processo finalizar no Servidor 3 o processo sai do sistema.

Deste ponto em diante o comportamento dos 3 servidores volta a ser semelhante. Após lidar com um processo finalizado, cada um zera sua variável interna de processo para *null* (*None* em Python), e faz a checagem novamente se há processos na fila. O comportamento é semelhante a primeira verificação da fila, com a exceção de que como neste caso o servidor estava ocupado, se existir um processo na fila desta vez o tempo de fila é levado em consideração e somado a variável interna de tempo de fila do servidor respectivo para o novo processo.

# 4 - Utilização

## 4.1 - Preparações

Definidas no início do *script* principal existem algumas variáveis globais para facilitar a execução das simulações, vamos entrar em detalhes sobre sua usabilidade:

- N_jobs: é um vetor de números inteiros onde cada posição representa uma simulação com n_jobs[i] processos executados no sistema
- Mode: é um vetor numérico onde cada posição representa uma opção de modo de amostragem de tempo de serviço conforme definido no item 2
- Mode_str: é um vetor de *strings* de mesmo tamanho que o vetor Mode, onde cada posição representa o nome do tipo de amostragem que faremos para o tempo de serviço dos processos, é utilizado para nomear arquivos de saída ao fim de cada experimento
- colunas: é um vetor de *strings* com nomes de cada métrica que queremos armazenar, é utilizado para facilitar a utilização de métodos da biblioteca Pandas
- TIME_STEP: em maiúsculo definido como constante, é a variável que rege o passo do nosso relógio discreto, neste caso 0.01 ou um centésimo de segundo

Após as definições de classes e variáveis globais a parte principal do programa ficou relativamente simples. Em pseudo código:

- Para cada valor n em n_jobs:
    - Para cada valor m em modes:
        - Criar Servidor 1 com mode m
        - Criar Servidor 2 com mode m
        - Criar Servidor 3 com mode m
        - Criar ou zerar vetor de tempos de chegada como especificado no item 3.2
        - Criar *Data Frame* global que guardará os tempos de cada processo
        - Criar ou zerar as filas Q1, Q2 e Q3 referentes a cada servidor e o contador do relógio discreto *currTime*

## 4.2 - Loop principal

Ainda dentro do laço modes, abaixo das preparações acima, a parte principal do programa e da simulação acontece:

- Repita Infinitamente:
    - Se o vetor de tempos de chegada não estiver vazio:
        - Se o tempo atual no relógio passou do tempo do primeiro elemento no vetor:
            - Adiciona na Fila do Servidor 1 o processo criado com valor de início de fila igual ao primeiro elemento no vetor de chegadas
            - Remove o primeiro elemento do vetor de chegadas
    - Passa uma unidade de tempo para o Servidor 1
    - Passa uma unidade de tempo para o Servidor 2
    - Passa uma unidade de tempo para o Servidor 3
    - Se o número de processos com resultados tabelados for maior que n:
        - Interrompa o Loop
    - Tempo Atual = Tempo Atual + Passo do Relógio
- Salva os resultados em um arquivo

# 5 - Resultados

Apliquei a simulação para 20 mil jobs conforme especificado, os valores diferem com frequência entre cada experimento. Mas seguem uma certa faixa. Como não obtive resultados muito confiáveis, conforme a recomendação do dia 11 de dezembro após a segunda prova, resolvi testar para 100 mil e 1 milhão de jobs também. Para 1 milhão a simulação ainda está rodando enquanto eu escrevo. Mas creio que os resultados de 20 mil e 100 mil serão satisfatórios por hora.

Foram removidos do cálculo da média e do desvio padrão a primeira metade das amostras para cada cenário como prevenção a representar o sistema fora de um estado de equilíbrio.

No geral, a média do caso onde os tempos são fixos variam entre 5 e 7, pelo que vi dos comentários dos colegas a moda parece ser variar de 6 a 8. Os resultados também são bem parecidos para o caso de tempo uniforme. Só no caso exponencial que temos um salto relevante tanto no tempo médio quanto no desvio padrão. Nos meus experimentos a média variou entre 11 e 14. O de alguns colegas variam de 12 a 15 raramente um 16 ou 17. O desvio padrão não é de surpreender que seja alto já que funções exponenciais possuem cauda muito longa no geral. Acredito que a discrepância entre os meus resultados e a

maioria dos colegas seja devido ao fato de eu ter utilizado o passo de tempo como um centésimo de segundo, mas não tenho provas.

| | Media | DesvioPadrao |
|---|---|---|
| *Fixo 20k* | 5.34487525630618 | 7.00429395138174 |
| *Fixo 100k* | 5.81339266618309 | 8.11212451525366 |
| *Fixo 1m* | 5.78378961909998 | 8.70581015969309 |
| *Uniforme 20k* | 5.84797185543633 | 6.89707737155127 |
| *Uniforme 100k* | 6.44793893411743 | 8.63977898747516 |
| *Exp 20k* | 11.805173207616 | 14.4848606461962 |
| *Exp 100k* | 13.6765455377075 | 21.7777956661079 |

Figura 2 - Tabela com os Resultados

# 6 - Conclusões

Foi muito bom poder implementar de fato o que vimos ao longo do período, consolidou bem os conceitos das distribuições e o fato de às vezes ser mais prático rodar uma simulação do que tentar descobrir uma distribuição exata que seja complicada. Além de ter sido uma ótima desculpa para revisar Orientação a Objetos. O código principal em Python se encontra no GitHub, junto com as tabelas CSV que resultaram nas métricas da tabela acima, este relatório, e um código em R para calcular as médias, os desvios padrões e gerar o gráfico da Figura 2, já que eu não tive coragem de encerrar o processo em Python que está rodando desde às duas da tarde (são dez da noite no momento).

# 7 - Repositório

https://github.com/GuilhermeVOS/Open_System_Queue_Simulation