Sistemas Distribuidos

*Balanceador de Carga para um Sistema Distribuído

1st Carlos Vasco Chironda
dept. Ciências da Computação
Universidade Estadual De Campinas
Campinas (SP) , Brasil
c298800@unicamp.br

2nd Márcio Levi Sales Prado
dept. Ciências da Computação
Universidade Estadual De Campinas
Campinas (SP) , Brasil
m183680@dac.unicamp.br

***Abstract*—Este documento descreve a implementação de um sistema distribuído com balanceador de carga, visando otimizar o uso de recursos e garantir eficiência na distribuição de requisições. O objetivo principal é desenvolver um balanceador que distribua a carga entre múltiplos servidores, evitando sobrecarga e subutilização, thereby promovendo escalabilidade e melhor alocação de recursos. Foram implementadas deversas políticas de balanceamento. O sistema simula tráfego real com requisições variadas (CPU-intensive e I/O-bound) que chegam em intervalos aleatórios, processadas por três servidores com capacidades similares. O desempenho foi avaliado através das métricas de *Throughput*, Tempo Médio de Resposta e Utilização do Sistema, permitindo analisar as condições ideais de aplicação para cada política em diferentes cenários de carga.**

I. Introduction

O crescimento exponencial da demanda por serviços computacionais na era digital tem tornado a eficiência na distribuição de carga um desafio crítico em sistemas distribuídos. A má alocação de recursos frequentemente resulta em situações onde alguns servidores operam sob sobrecarga enquanto outros permanecem subutilizados, levando a degradação do desempenho, aumento de latência e má experiência do usuário.

Este projeto visa implementar e analisar um sistema distribuído com balanceador de carga desenvolvido em Python utilizando a biblioteca SimPy [1] para simulação de eventos discretos. O trabalho foca na implementação e comparação de oito políticas de balanceamento de carga:

- **Políticas Básicas:** Random, Round Robin, Shortest Queue [2]
- **Políticas Baseadas em Métricas:** Least Response Time, Least Connections [3]
- **Políticas Avançadas:** Power of Two Choices (P2C) Least Load, Least Expected Delay (LED) [4]
- **Política Adaptativa:** Adaptive

A escolha do SimPy como framework de simulação justifica-se pela sua eficiência em modelar sistemas discretos com componentes concorrentes, permitindo a simulação realista de tráfego de rede, processamento de requisições e comportamento assíncrono dos servidores.

Identify applicable funding agency here. If none, delete this.

O sistema simulado consiste em um balanceador de carga que distribui requisições entre múltiplos servidores, com tráfego variado que inclui requisições CPU-intensive, I/O-bound e mistas, chegando em padrões aleatórios que imitam cenários do mundo real. Foram implementados mecanismos para monitoramento em tempo real do estado dos servidores, comprimento das filas e métricas de desempenho.

A principal contribuição deste trabalho está na análise comparativa abrangente entre políticas tradicionais e avançadas, identificando os cenários específicos onde cada estratégia demonstra superioridade em termos de throughput, tempo médio de resposta e utilização de recursos. Esta análise fornece insights valiosos para arquitetos de sistemas na seleção da política mais adequada para diferentes cargas de trabalho e requisitos de qualidade de serviço.

A estrutura deste relatório apresenta na Seção 2 a estrutura do servidor ,Seção 3 os fundamentos teóricos das políticas implementadas, na Seção 4 a arquitetura do sistema e metodologia de implementação, na Seção 5 os resultados experimentais e análise comparativa, e finalmente nas Seções 6 e 7 as conclusões e trabalhos futuros.

II. Servidor

Em um ambiente real de um sistema distribuído, os servidores são a componente principal, pois fornecem os serviços requeridos na rede distribuída. Neste nosso projeto para simulação implementamos servidores com a biblioteca SimPy.

A classe `Servidor` foi desenvolvida para simular o comportamento de servidores reais, utilizando `simpy.Store` para gerenciar uma fila explícita de requisições. Cada servidor possui parâmetros configuráveis como capacidade da fila, velocidade de processamento e um fator *alpha* para cálculo EMA (*Exponential Moving Average*) do tempo de resposta. O processamento diferencia requisições CPU-intensive e I/O-bound, com tempos baseados em distribuições exponenciais. O servidor monitora métricas essenciais como taxa de ocupação, número de requisições processadas e tempo médio de resposta em tempo real.

III. Balanceadores

Os balanceadores representam o coração do desenvolvimento desse trabalho, pois o objetivo fundamental é projetar

um sistema distribuído que otimiza os recursos e que seja eficiente. No nosso caso os balanceadores foram implementados utilizando a classe *Balanceador* e cada função dessa classe representa a implementação de um balanceador, que são os seguintes:

A. Random

Seleciona aleatoriamente qualquer servidor disponível no pool. É simples de implementar e não requer monitoramento de estado, mas pode levar a distribuição desigual em cenários com cargas heterogêneas. Adequado para ambientes com servidores idênticos e carga uniforme.

B. Random

Seleciona aleatoriamente qualquer servidor disponível no pool. É simples de implementar e não requer monitoramento de estado, mas pode levar a distribuição desigual em cenários com cargas heterogêneas. Adequado para ambientes com servidores idênticos e carga uniforme.

C. Round Robin

Distribui as requisições sequencialmente entre os servidores, garantindo distribuição equitativa independentemente da carga atual. Mantém um índice circular para rotação sistemática. Ideal para servidores homogêneos com capacidades similares e carga previsível.

D. Shortest Queue

Encaminha a requisição para o servidor com menor número de tarefas na fila. Requer monitoramento contínuo do tamanho das filas. Eficaz para minimizar tempo de espera quando as requisições têm durações similares.

E. Least Response Time

Seleciona o servidor com menor tempo médio de resposta histórico, usando média móvel exponencial (EMA) para suavizar variações. Adapta-se dinamicamente ao desempenho real dos servidores, preferindo os mais rápidos.

F. Least Connections

Escolhe o servidor com menor número de conexões ativas, considerando tanto a fila quanto o estado ocupado. Similar ao Shortest Queue mas inclui a requisição em processamento. Bom para cargas com durações variáveis.

G. Power of Two Choices (p2c)

Seleciona aleatoriamente dois servidores e escolhe o com menor carga atual. Combina aleatoriedade com seleção informada, oferecendo bom balanceamento com baixo overhead. Compensa a simplicidade do Random com eficiência próxima do ótimo.

H. Least Load

Encaminha para o servidor com menor tempo acumulado de ocupação, normalizado pelo tempo total de simulação. Prefere servidores menos utilizados historicamente, promovendo distribuição de carga de longo prazo.

I. Least Expected Delay (LED)

Calcula o delay esperado baseado no tamanho da fila, estado do servidor e sua velocidade. Minimiza o tempo total previsto de processamento. Considera tanto a carga atual quanto a capacidade do servidor.

J. Adaptive

Alterna entre políticas conforme a carga do sistema: usa Random em carga baixa e Shortest Queue em carga alta. Adapta-se dinamicamente às condições do sistema, otimizando para diferentes cenários de tráfego.

IV. Geração de Requisições e Simulação

Para simular cenários realistas, implementou-se um gerador de requisições que produz cargas de trabalho variadas. O sistema gera requisições do tipo CPU-bound e I/O-bound com tamanhos aleatórios entre 1 e 5 unidades, seguindo uma distribuição exponencial para os tempos de chegada.

A função `simular()` coordena toda a execução, inicializando o ambiente SimPy, configurando três servidores com capacidades escalonadas (1x, 2x, 4x) e o balanceador com a política desejada. Durante a simulação, são coletadas métricas essenciais como *throughput*, tempo médio de resposta e utilização dos servidores, permitindo a comparação objetiva do desempenho de cada política sob condições controladas.

V. Resultados

Como já citado no início, para poder analisar as prestações do nosso sistema utilizamos as seguintes métricas:

1) **Throughput**: Número de requisições processadas por unidade de tempo
2) **Tempo Médio de Resposta**: Tempo total desde a chegada até o processamento completo
3) **Utilização**: Percentagem de tempo que os servidores permanecem ocupados

A. Análise Detalhada das Políticas Principais

1) Random: A política **Random** demonstrou um desempenho consistente em baixas cargas (até taxa 0.50), com throughput de 0.4906 e tempo médio de 8.9651. No entanto, à medida que a carga aumenta, o desempenho degrada significativamente. A partir da taxa 0.60, o tempo médio de resposta dispara para 24.51 e continua crescendo exponencialmente, atingindo 2015.85 na taxa 10.00. A utilização mantém-se elevada (1.9986 na taxa máxima), indicando que os servidores estão constantemente ocupados, mas a distribuição aleatória leva a desbalanceamento severo sob alta carga.

2) Round Robin: O **Round Robin** mostrou-se ligeiramente superior ao Random em cargas moderadas, com melhor tempo médio de resposta (5.5173 vs 6.4812 na taxa 0.40). Contudo, sob cargas elevadas, apresenta problemas similares, com tempo médio chegando a 2023.71 na taxa 10.00. A distribuição equitativa previne o completo colapso de servidores individuais, mas não considera a capacidade de processamento, resultando em ineficiências quando os servidores têm capacidades diferentes.

3) Shortest Queue: A política **Shortest** revelou-se notavelmente superior sob altas cargas. Enquanto as outras políticas sofrem degradação catastrófica do tempo de resposta, o Shortest mantém tempos consistentes entre 6-8 unidades até a taxa 1.59, e mesmo sob carga extrema (taxa 10.00) mantém 452.95 vs 2015+ das outras. O throughput também é significativamente melhor, atingindo 1.0086 na taxa 0.99 vs 0.8902 do Random.

B. Análise Resumida das Demais Políticas

1) Power of Two Choices (p2c): Apresenta excelente equilíbrio entre simplicidade e eficiência. Em baixas cargas, tem throughput competitivo (0.2062 na taxa 0.20) e bom tempo de resposta (5.1888). Sob altas cargas, mantém desempenho estável com throughput de 1.5392 e tempo de resposta 2061.68 na taxa 10.00, superando significativamente as políticas básicas.

2) Least Load: Boa performance em cargas moderadas, mas sofre sob cargas elevadas. Na taxa 1.59, o tempo de resposta atinge 178.86, demonstrando dificuldade em lidar com picos de carga. Porém, mantém utilização mais equilibrada entre servidores.

3) Least Expected Delay (LED): Destaca-se pelo excelente tempo de resposta em cargas baixas e moderadas (4.18 na taxa 0.69). Mesmo sob carga total (taxa 10.00), mantém tempo de resposta em 1338.65, sendo uma das melhores opções para aplicações sensíveis à latência.

4) Adaptive: Mostra versatilidade, adaptando-se bem a diferentes condições de carga. Performance consistente across do espectro, com throughput de 1.5762 e tempo de resposta 429.93 na taxa máxima, demonstrando boa capacidade de ajuste dinâmico.

5) Least Response Time: Excelente em cargas baixas (tempo de resposta de apenas 2.51 na taxa 0.20), mas degrada rapidamente sob alta carga, atingindo 1370.55 na taxa 10.00. Ideal para ambientes com carga previsível e moderada.

6) Least Connections: Comportamento similar ao Shortest Queue, com boa performance geral. Na taxa 10.00, atinge throughput de 1.5506 e tempo de resposta 471.66, demonstrando eficácia na distribuição baseada no estado atual das conexões.

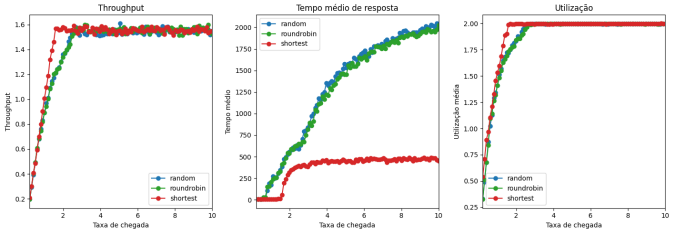

Fig. 1. Comparação das métricas entre as políticas de balanceamento [8]

C. Análise Comparativa Geral

Os resultados demonstram claramente a superioridade de políticas baseadas em estado do sistema (Shortest Queue, Least Connections, LED) sobre as estáticas (Random, Round Robin) sob cargas elevadas. As políticas adaptativas (Adaptive, p2c) oferecem bom equilíbrio entre complexidade e performance.

O **Least Expected Delay (LED)** emerge como a melhor opção geral para aplicações que requerem baixa latência, enquanto o **Shortest Queue** é ideal para maximizar throughput em ambientes sob alta carga. Para implementações que buscam equilíbrio entre simplicidade e eficácia, o **Power of Two Choices (p2c)** representa excelente opção.

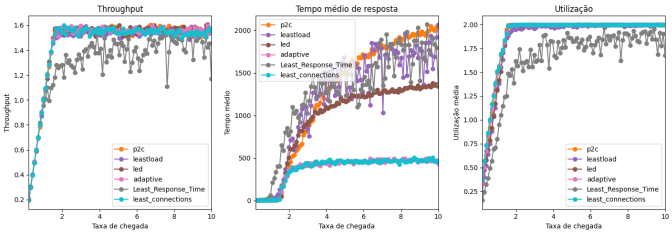

Fig. 2. Comparação das métricas entre p2c, LD, LED,AdA,LST [8]

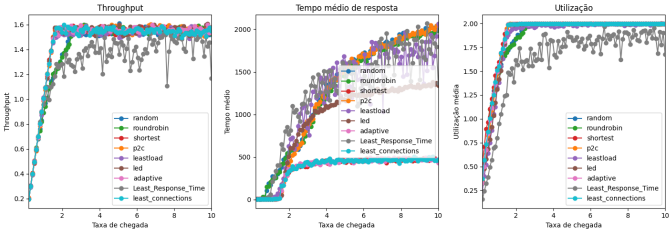

Fig. 3. Comparação das métricas de todos os balanceadores [8]

VI. Conclusão

Este trabalho implementou e analisou oito políticas de balanceamento de carga em sistemas distribuídos, demonstrando que a escolha da estratégia adequada impacta significativamente no desempenho do sistema. As políticas baseadas em estado do sistema (Shortest Queue, Least Connections, LED) mostraram-se superiores sob altas cargas, enquanto as estáticas (Random, Round Robin) são adequadas apenas para cenários de carga leve. O Power of Two Choices emergiu como excelente equilíbrio entre simplicidade e eficiência. Os resultados comprovam que monitoramento em tempo real do estado dos servidores é essencial para otimização de recursos em ambientes distribuídos.

VII. Trabalhos Futuros

Como extensões deste trabalho, propõe-se: implementação de políticas híbridas que combinem múltiplas estratégias, adaptação para ambientes de nuvem heterogêneos, inclusão de mecanismos de previsão de carga baseados em machine learning, e extensão para suportar balanceamento em múltiplas camadas (aplicação, banco de dados, cache). Adicionalmente, seria valioso testar as políticas em ambientes de produção com cargas reais.

REFERENCES

[1] Team SimPy, “SimPy: Discrete event simulation for Python”, Disponível: https://simpy.readthedocs.io/, 2023.

[2] V. Cardellini, M. Colajanni, P. S. Yu, “Dynamic Load Balancing on Web-Server Systems”, IEEE Internet Computing, vol. 3, no. 3, pp. 28-39, 1999

[3] A. N. Tantawi, D. Towsley, “Optimal static load balancing in distributed computer systems”, Journal of the ACM, vol. 32, no. 2, pp. 445-465, 1985.

[4] M. Mitzenmacher, “The Power of Two Choices in Randomized Load Balancing”, IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 10, pp. 1094-1104, 2001.

[5] P. B. S. S. K. R. M. K. G. S. K. Das, “Adaptive load balancing algorithms in distributed systems”, Journal of Parallel and Distributed Computing, vol. 115, pp. 84-97, 2018.

[6] J. D. C. Little, “A Proof for the Queuing Formula: L = W”, Operations Research, vol. 9, no. 3, pp. 383-387, 1961.

[7] G. van Rossum et al., “Python Language Reference”, Disponível: https://www.python.org/, 2023.

[8] J. D. Hunter, “Matplotlib: A 2D Graphics Environment”, Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.