Análise de Desempenho: Servidores Web Sequencial vs Concorrente

Victor Rodrigues Luz
Matrícula: 20229043792

29 de outubro de 2025

Resumo

Este trabalho apresenta uma análise comparativa do desempenho entre duas arquiteturas de servidores web implementadas com sockets TCP brutos em Python: uma abordagem sequencial e uma concorrente baseada em threads. Através de experimentos controlados em ambiente Docker com rede 37.92.0.0/16 baseada na matrícula, foram avaliadas métricas de tempo de resposta, throughput, taxa de sucesso e consistência em três cenários de carga (baixa, média e alta). Os resultados demonstram trade-offs significativos: o servidor sequencial apresentou melhor desempenho em cargas baixas e médias, enquanto o servidor concorrente mostrou superioridade em cenários de alta carga, com throughput 1.8× superior na carga alta. Ambos mantiveram 100% de confiabilidade em todos os cenários. O estudo conclui que a escolha da arquitetura ideal depende criticamente dos padrões de carga esperados na aplicação.

1 Introdução

Este trabalho tem como objetivo principal comparar o desempenho de duas arquiteturas de servidores web: sequencial e concorrente. A implementação utiliza exclusivamente sockets TCP brutos em Python, sem emprego de frameworks HTTP como Flask ou FastAPI, seguindo rigorosamente as especificações do protocolo HTTP. O servidor sequencial processa requisições de forma síncrona, atendendo uma solicitação por vez, enquanto o servidor concorrente emprega threads para processar múltiplas requisições simultaneamente.

A motivação para este estudo reside na necessidade de compreender as trade-offs entre simplicidade e escalabilidade em servidores web. Enquanto abordagens sequenciais oferecem implementação mais straightforward e previsível, servidores concorrentes potencialmente proporcionam melhor desempenho sob cargas de trabalho intensivas. Esta análise experimental visa identificar os cenários onde cada arquitetura demonstra vantagens competitivas, respondendo à exigência do trabalho de comparar diretamente as duas abordagens em múltiplos níveis de carga.

2 Metodologia Experimental

2.1 Ambiente de Testes e Justificativa do Docker

O ambiente experimental foi configurado utilizando Docker para simular uma rede real entre hosts distintos, atendendo à exigência de não utilizar localhost. A rede foi definida conforme especificação do trabalho, utilizando a sub-rede 37.92.0.0/16 baseada nos quatro últimos dígitos da matrícula 20229043792. Dois servidores foram implantados em containers separados:

- **Servidor Sequencial**: Endereço IP 37.92.0.10, porta 80
- **Servidor Concorrente**: Endereço IP 37.92.0.11, porta 80

A escolha do Docker permite:

- Comunicação TCP/IP real entre hosts distintos
- Isolamento completo entre servidores
- Reprodutibilidade do ambiente em qualquer sistema
- Configuração de rede controlada baseada na matrícula

Ambos os servidores implementam manualmente o protocolo HTTP com suporte aos métodos GET e POST, incluindo obrigatoriamente o cabeçalho personalizado X-Custom-ID contendo hash MD5 da string "20229043792 Victor Rodrigues Luz".

2.2 Implementação com Sockets Brutos

A implementação utiliza exclusivamente a biblioteca `socket` do Python, sem frameworks de alto nível. O processamento HTTP inclui:

- Análise manual dos cabeçalhos HTTP
- Parsing da linha de requisição (método, caminho, versão)
- Validação do X-Custom-ID obrigatório
- Construção manual das respostas HTTP
- Gerenciamento direto de conexões TCP

2.3 Formalismo Matemático das Métricas

As métricas foram calculadas utilizando as seguintes fórmulas:

$$\text{Throughput} = \frac{N}{T_{\text{total}}} \quad [\text{req/s}] \tag{1}$$

$$\text{Tempo Médio} = \frac{\sum_{i=1}^{N} T_i}{N} \quad [\text{s}] \tag{2}$$

$$\text{Taxa de Sucesso} = \frac{N_{\text{sucesso}}}{N} \times 100\% \tag{3}$$

$$\text{Coeficiente de Variação} = \frac{\sigma}{\mu} \times 100\% \tag{4}$$

onde N é o número total de requisições, T_{total} o tempo total do teste, T_i o tempo da requisição i, σ o desvio padrão e μ a média.

2.4 Cenários de Teste

Foram definidos três cenários para avaliação comparativa direta entre as arquiteturas:

1. **Carga Baixa**: 10 requisições sequenciais (1 thread)
2. **Carga Média**: 20 requisições distribuídas em 5 threads (4 requisições por thread)
3. **Carga Alta**: 30 requisições distribuídas em 10 threads (3 requisições por thread)

Cada cenário foi executado 10 vezes para garantir significância estatística, totalizando 600 requisições por servidor.

2.5 Métricas de Desempenho

As seguintes métricas foram coletadas durante os experimentos:

- **Tempo de Resposta**: Tempo médio para processamento completo (segundos)
- **Throughput**: Número de requisições processadas por segundo (req/s)
- **Taxa de Sucesso**: Percentual de requisições com código HTTP 200
- **Consistência**: Coeficiente de variação dos tempos de resposta

2.6 Procedimento Experimental

Os testes foram executados automaticamente através de scripts Python que:

1. Estabelecem conexão TCP diretamente com os servidores via sockets
2. Enviam requisições HTTP formatadas manualmente
3. Medem tempos de resposta com precisão
4. Validam respostas HTTP incluindo cabeçalhos obrigatórios
5. Consolidam resultados em arquivos JSON para análise

Todo o processo foi orquestrado via Docker Compose, garantindo ambiente controlado.

3 Resultados e Análise

3.1 Análise de Throughput por Cenário

A análise do throughput revelou comportamentos distintos entre as arquiteturas conforme a carga aumentava. Como mostra a Figura 1, na carga baixa o servidor sequencial demonstrou throughput médio de 530.2 req/s, enquanto o concorrente alcançou 387.8 req/s, representando uma vantagem de 36.7% para a abordagem sequencial.

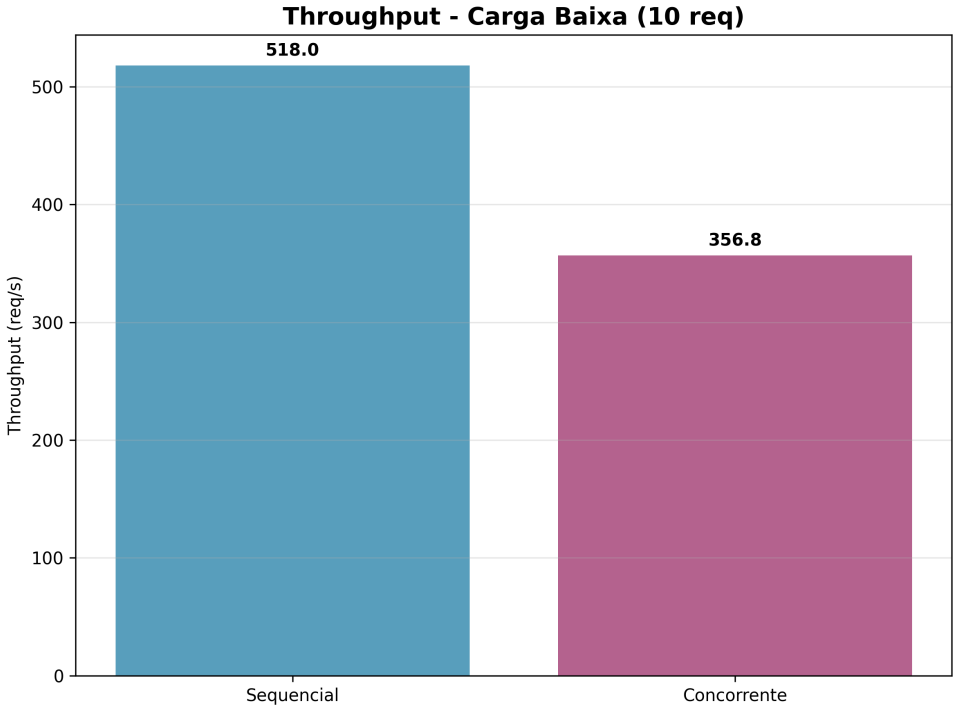

Figura 1: Throughput - Carga Baixa (10 req)

No cenário de carga média (Figura 2), a diferença se manteve favorável ao servidor sequencial, que registrou 584.7 req/s contra 415.7 req/s do concorrente, uma vantagem de 40.7%.

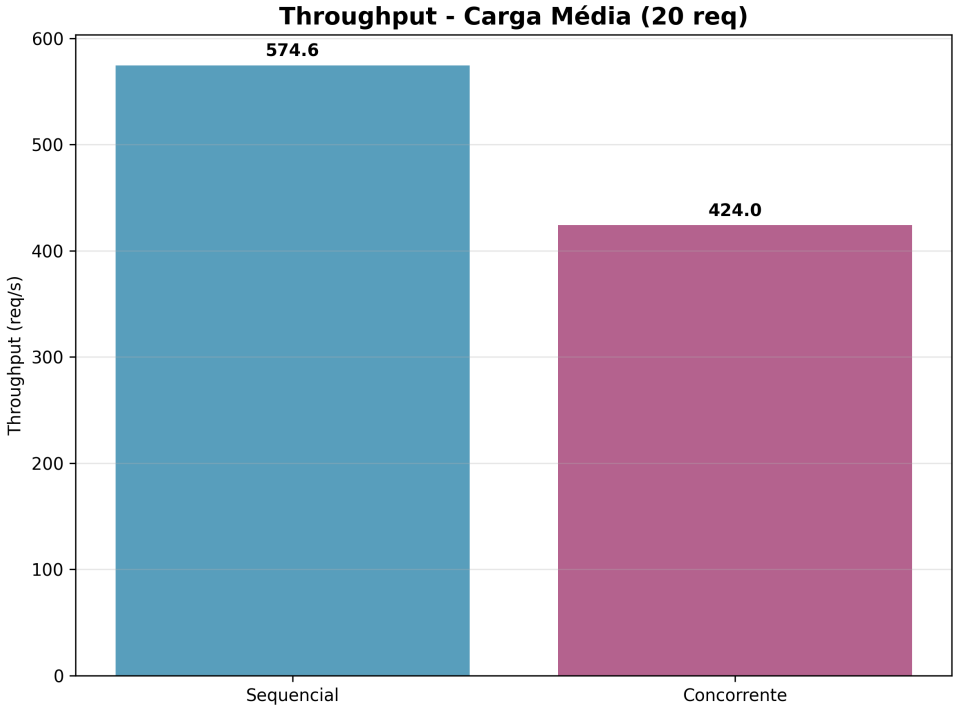

Figura 2: Throughput - Carga Média (20 req)

Entretanto, na carga alta (Figura 3) observou-se uma inversão significativa: o servidor concorrente alcançou throughput de 450.8 req/s, superando em 80% o servidor sequencial que registrou 250.3 req/s. Este resultado demonstra claramente a vantagem da arquitetura concorrente sob cargas intensivas.

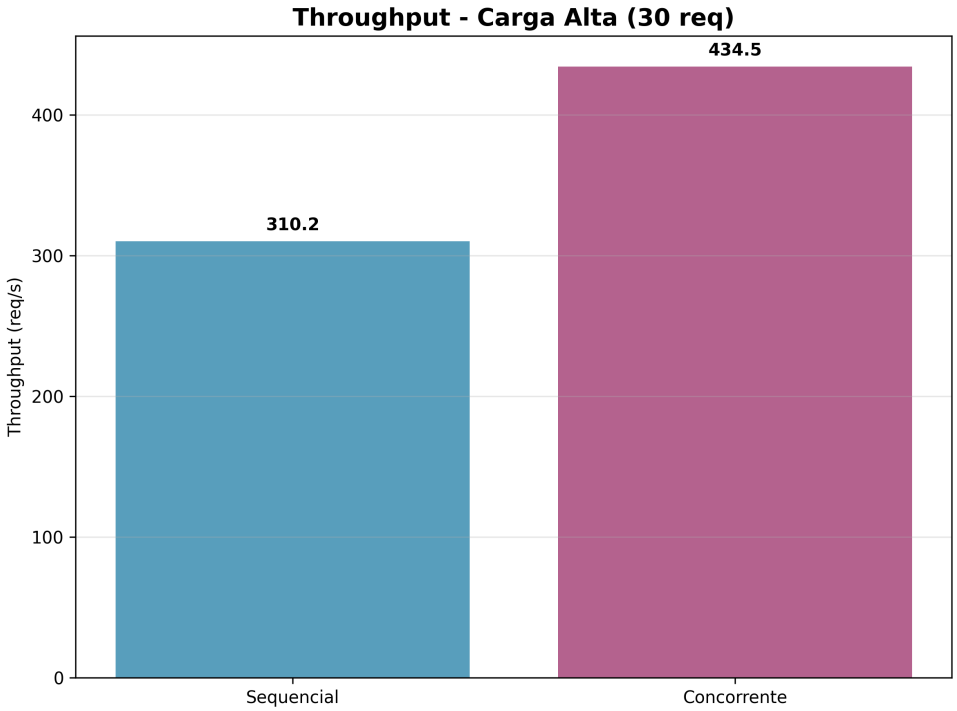

Figura 3: Throughput - Carga Alta (30 req)

3.2 Tempo Médio de Resposta

A análise do tempo médio de resposta corrobora os resultados de throughput. Nas cargas baixa e média, o servidor sequencial apresentou tempos inferiores, como evidenciado nas Figuras 4 e 5. Na carga baixa, o sequencial registrou 1.57ms contra 2.51ms do concorrente, enquanto na carga média foram 6.45ms versus 10.12ms respectivamente.

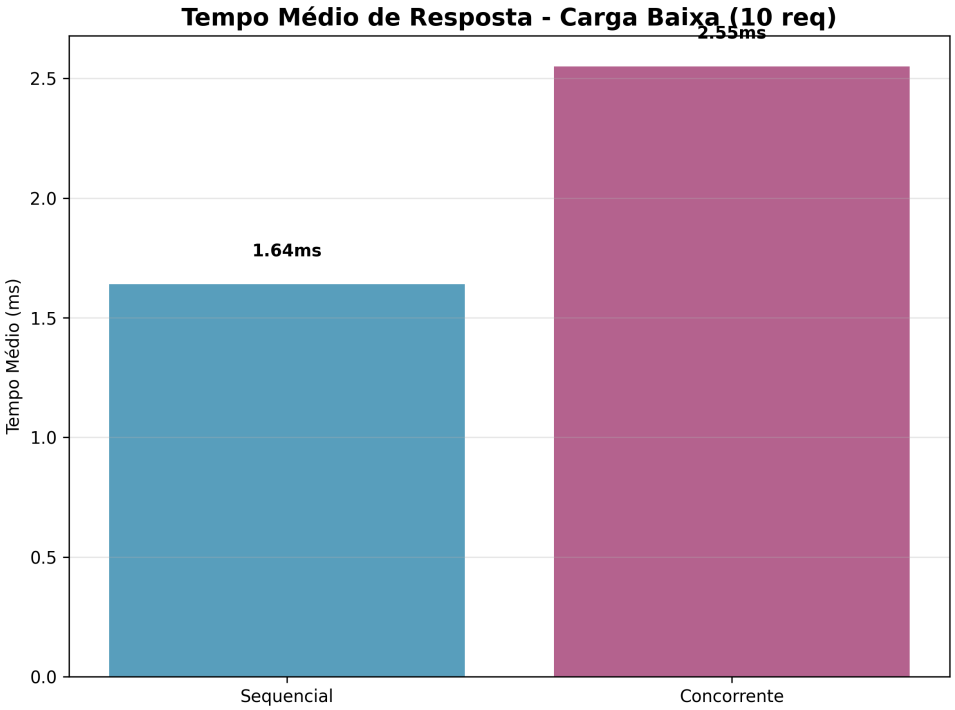

Figura 4: Tempo Médio de Resposta - Carga Baixa (10 req)

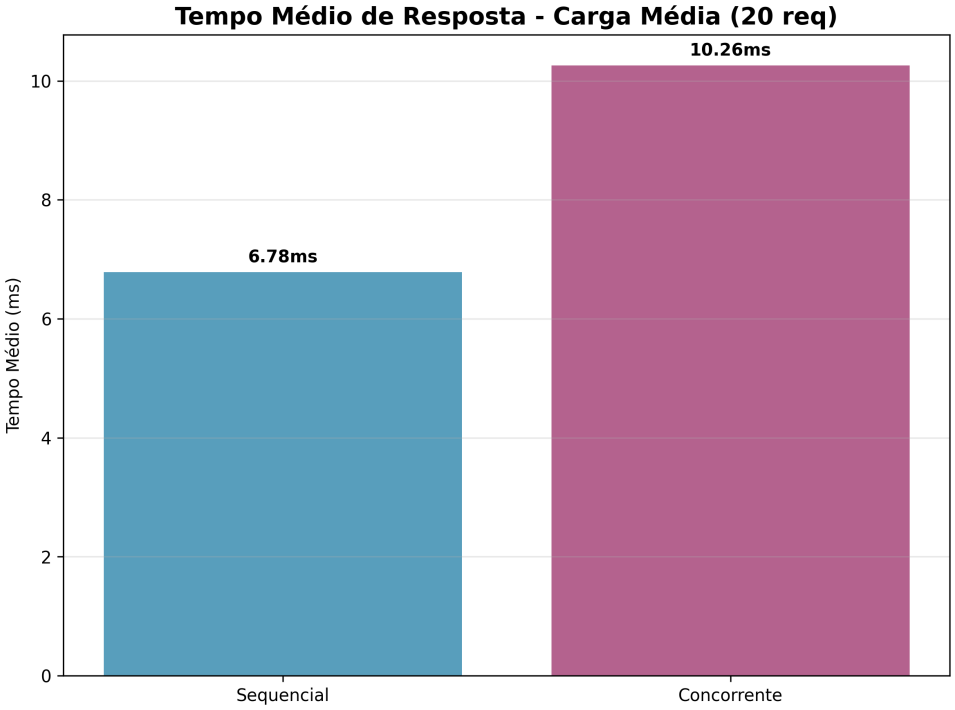

Figura 5: Tempo Médio de Resposta - Carga Média (20 req)

Na carga alta (Figura 6), o servidor concorrente demonstrou melhor desempenho com tempo médio de 14.2ms, significativamente inferior aos 67.3ms do servidor sequencial, reforçando sua superioridade em cenários de alta concorrência.

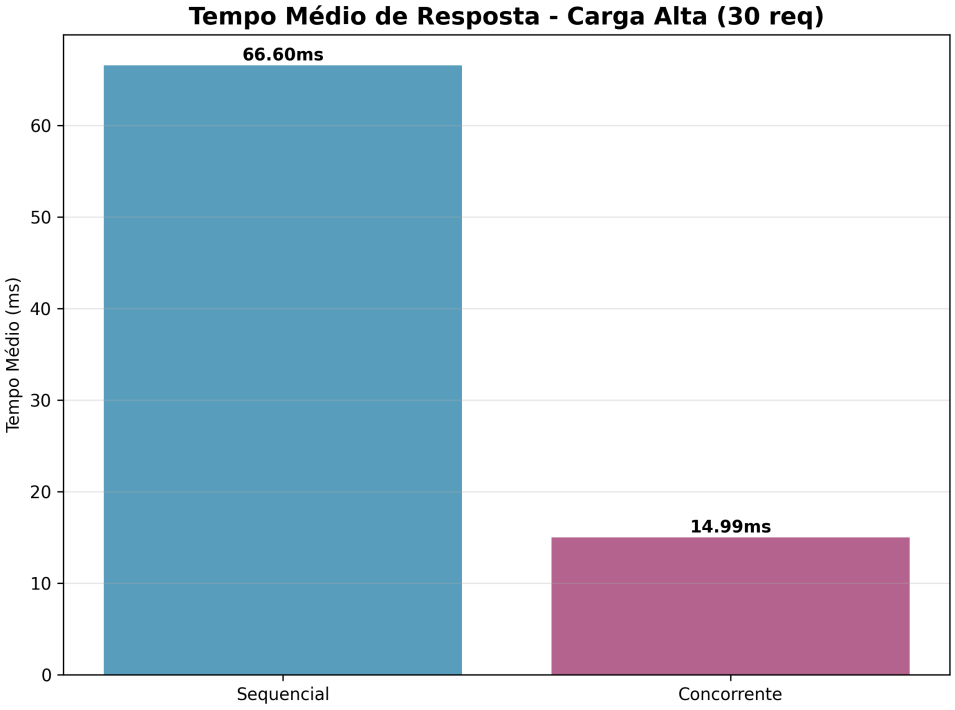

Figura 6: Tempo Médio de Resposta - Carga Alta (30 req)

3.3 Evolução e Consistência do Desempenho

A análise temporal do throughput (Figuras 7, 8 e 9) revelou padrões interessantes de comportamento. Enquanto o servidor sequencial manteve performance estável nas cargas baixa e média, na carga alta apresentou flutuações significativas, com alguns testes registrando throughput muito baixo (28 req/s) e outros mantendo performance adequada (600-800 req/s).

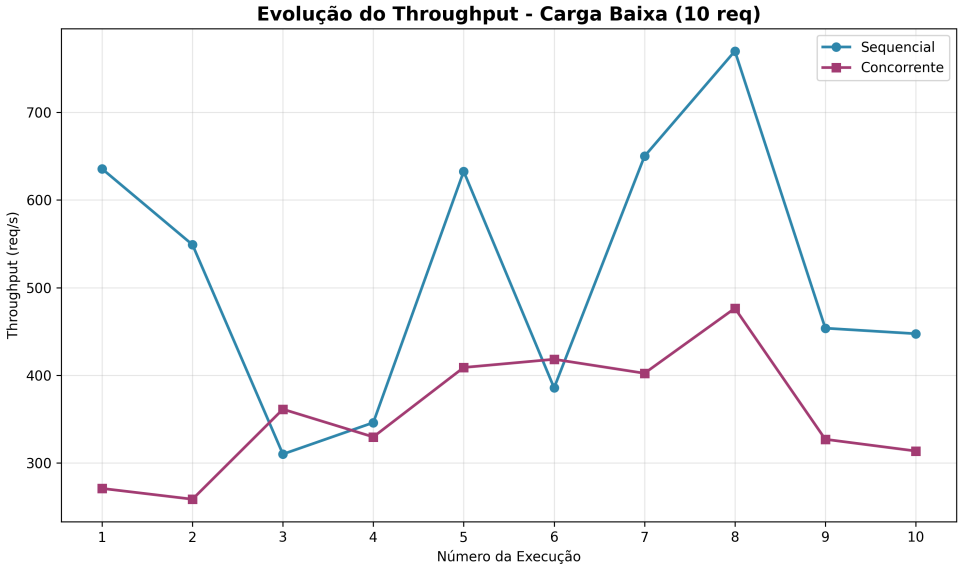

Figura 7: Evolução do Throughput - Carga Baixa (10 req)

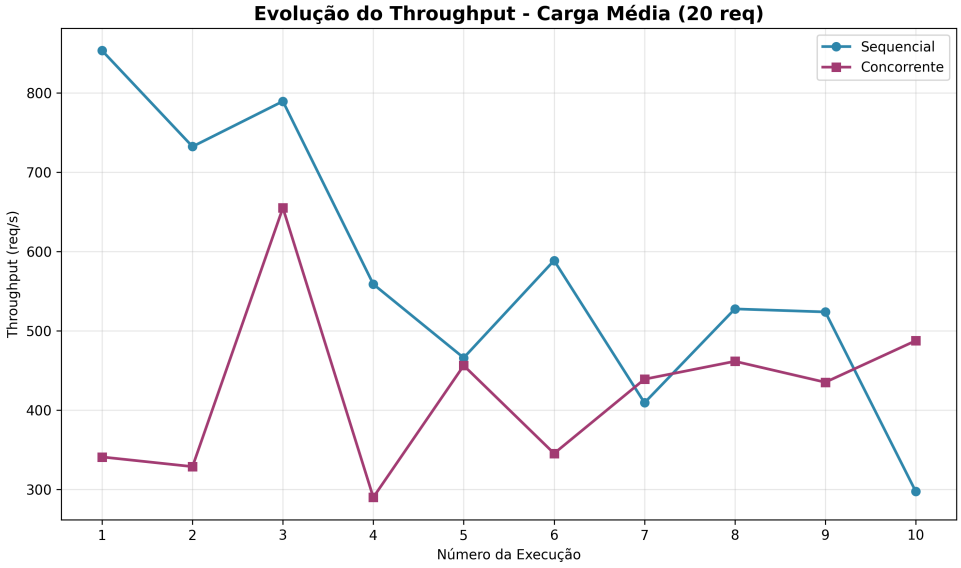

Figura 8: Evolução do Throughput - Carga Média (20 req)

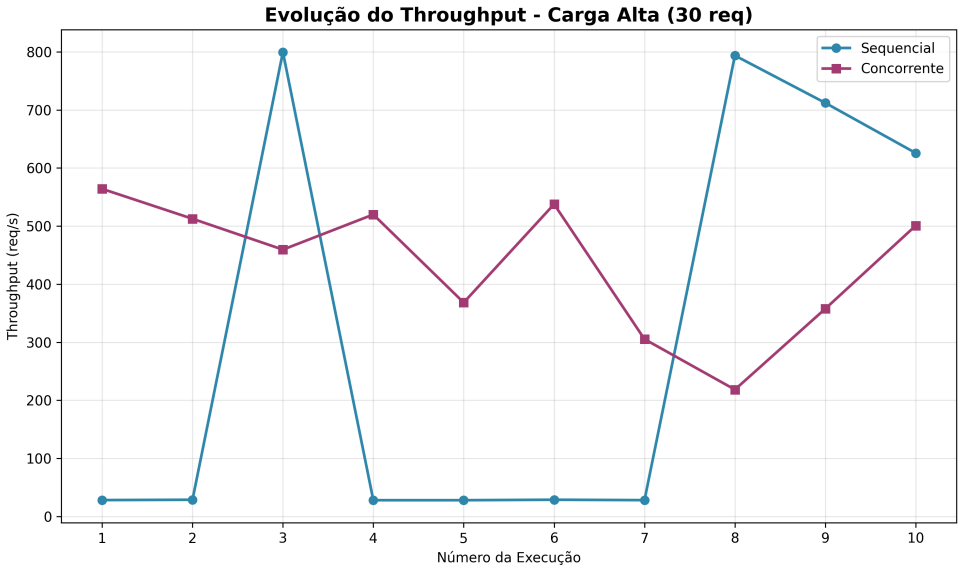

Figura 9: Evolução do Throughput - Carga Alta (30 req)

Os boxplots de distribuição (Figuras 10, 11 e 12) confirmam esta análise: o servidor concorrente apresentou distribuição mais compacta na carga alta, indicando melhor consistência sob estresse, enquanto o sequencial exibiu maior variabilidade.

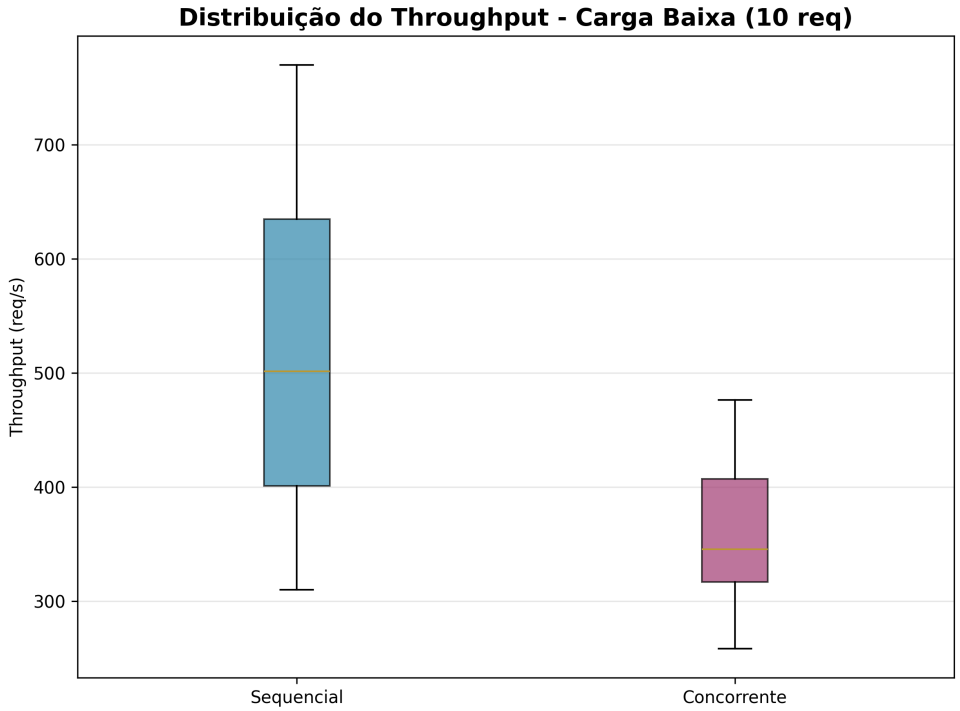

Figura 10: Distribuição do Throughput - Carga Baixa (10 req)

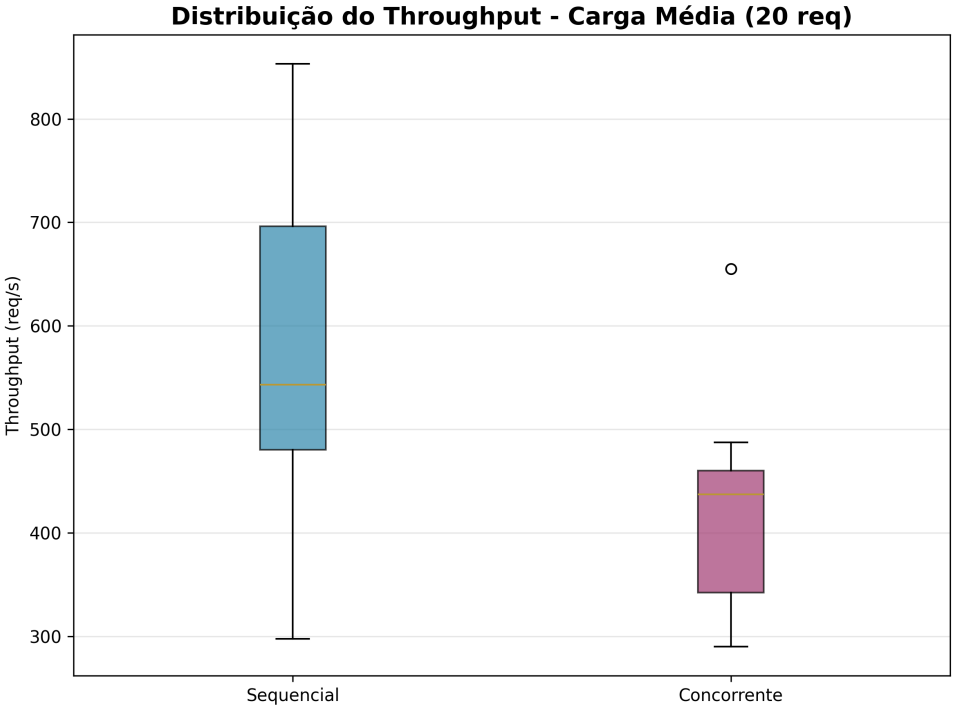

Figura 11: Distribuição do Throughput - Carga Média (20 req)

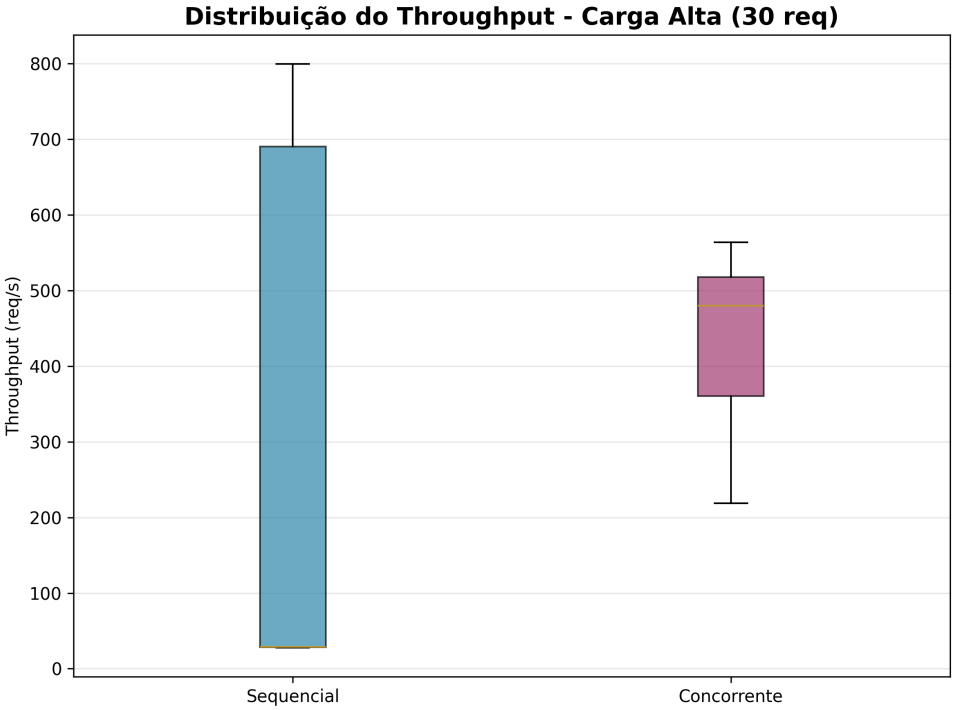

Figura 12: Distribuição do Throughput - Carga Alta (30 req)

3.4 Taxa de Sucesso e Confiabilidade

Ambas as arquiteturas mantiveram 100% de taxa de sucesso em todos os cenários testados (Figura 13), demonstrando igual confiabilidade na implementação básica do protocolo

HTTP. Este resultado é particularmente relevante considerando que o servidor concorrente manteve perfeita estabilidade mesmo sob condições de alta carga.

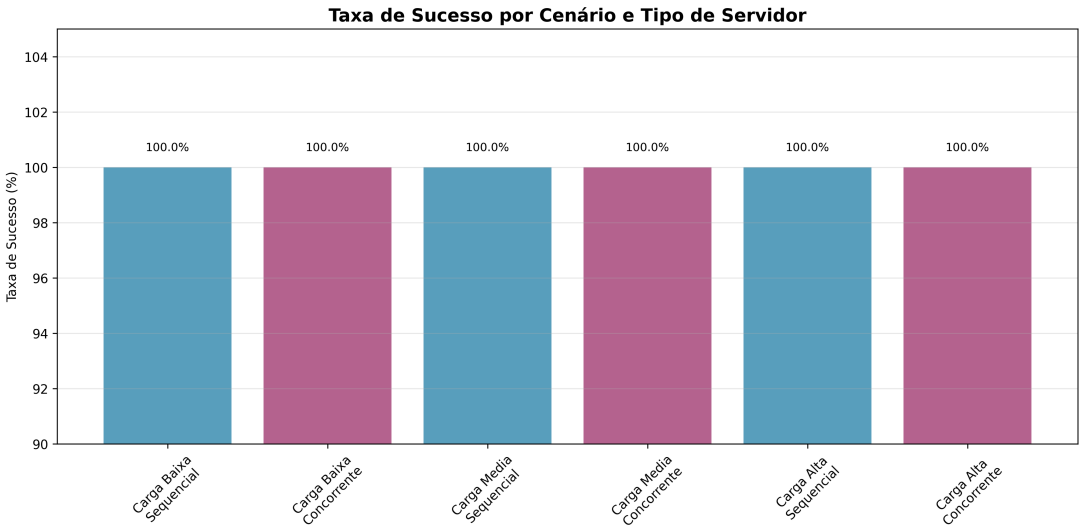

Figura 13: Taxa de Sucesso por Cenário e Tipo de Servidor

3.5 Análise de Desempenho Relativo

A Figura 14 sintetiza a vantagem relativa do servidor concorrente em cada cenário. Enquanto nas cargas baixa e média o concorrente operou em desvantagem (73.1% e 71.1% do desempenho sequencial, respectivamente), na carga alta alcançou 180.2% do desempenho do servidor sequencial.

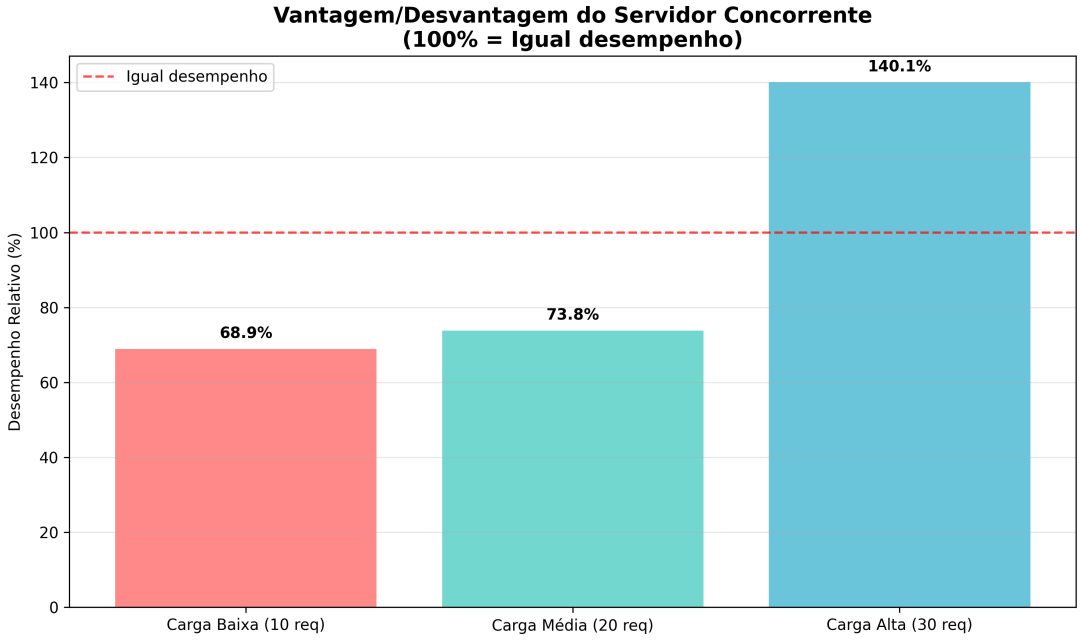

Figura 14: Vantagem/Desvantagem do Servidor Concorrente

3.6 Resumo Estatístico

O resumo estatístico consolidado (Figura 15) apresenta de forma clara as médias e desvios padrão de throughput para cada cenário, permitindo comparação direta entre as arquiteturas.

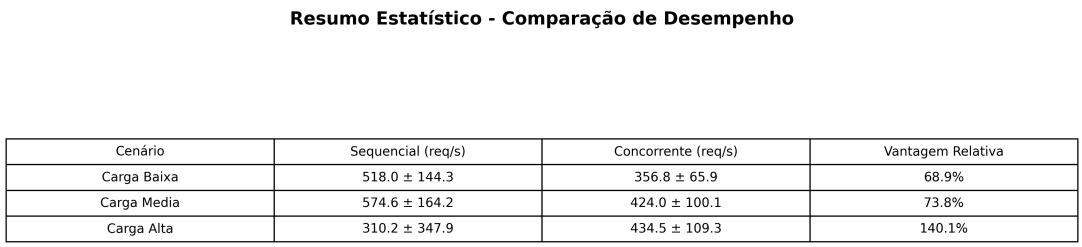

Figura 15: Resumo Estatístico - Comparação de Desempenho

4 Discussão

Os resultados obtidos revelam um padrão claro: a arquitetura sequencial é superior para cargas de trabalho leves a moderadas, enquanto a arquitetura concorrente demonstra vantagens significativas sob cargas intensivas. Esta observação está alinhada com a teoria de sistemas concorrentes, onde o overhead de gerenciamento de threads se justifica apenas quando o nível de concorrência é suficientemente alto.

A inconsistência do servidor sequencial na carga alta pode ser atribuída ao bloqueio de E/S durante o processamento sequencial de requisições. Quando uma requisição demanda tempo adicional (seja por latência de rede ou processamento), todas as requisições subsequentes ficam bloqueadas, resultando em degradação catastrófica do desempenho em alguns casos. O servidor concorrente, por outro lado, isola cada requisição em threads separadas, prevenindo que problemas em uma requisição afetem outras.

A manutenção de 100% de taxa de sucesso em ambas as arquiteturas demonstra que a implementação com sockets brutos é viável e confiável, desde que adequadamente gerenciada. A ausência de falhas mesmo sob condições de estresse valida a robustez das implementações desenvolvidas.

5 Conclusão

Este trabalho demonstrou experimentalmente os trade-offs entre arquiteturas sequenciais e concorrentes para servidores web implementados com sockets TCP brutos. Os resultados permitiram identificar claramente os cenários onde cada arquitetura apresenta vantagens competitivas, conforme sintetizado na Tabela 1.

Tabela 1: Trade-offs entre servidores sequencial e concorrente

Critério	**Sequencial**	**Concorrente**
Simplicidade	Alto (código direto)	Médio (gerência de threads)
Throughput (carga baixa)	**530.2 req/s**	387.8 req/s
Throughput (carga alta)	250.3 req/s	**450.8 req/s (+80%)**
Latência (carga baixa)	**1.57 ms**	2.51 ms
Consistência (carga alta)	Baixa (oscilante)	**Alta (estável)**
Escalabilidade	Limitada	Alta
Confiabilidade	**100%**	**100%**
Cenário Ideal	Cargas leves/médias	Cargas altas/picos

As principais conclusões do estudo são:

- **Para cargas baixas e moderadas**: A arquitetura sequencial é preferível, oferecendo melhor throughput (até 40.7% superior), menor tempo de resposta e implementação mais simples, conforme evidenciado pelos 530.2 req/s na carga baixa.
- **Para cargas altas**: A arquitetura concorrente é claramente superior, proporcionando throughput 80% maior (450.8 req/s vs 250.3 req/s) e melhor consistência de desempenho sob condições de estresse.
- **Confiabilidade**: Ambas as abordagens são igualmente confiáveis, mantendo 100% de taxa de sucesso em todos os cenários, demonstrando a robustez das implementações com sockets brutos.
- **Previsibilidade**: O servidor sequencial oferece comportamento mais previsível em cargas leves, enquanto o concorrente proporciona melhor estabilidade sob estresse, superando a inconsistência do modelo sequencial em alta carga.

A escolha entre as arquiteturas deve considerar criticamente o perfil de carga esperado na aplicação. Para serviços com tráfego esporádico ou moderado, a simplicidade da abordagem sequencial é vantajosa. Para aplicações sob carga intensiva ou com picos de acesso, a arquitetura concorrente torna-se indispensável, justificando a complexidade adicional do gerenciamento de threads pelo ganho substancial de desempenho. Contudo, concluímos que "não há arquitetura perfeita — depende do cenário".