Tópicos Especias em Engenharia de Software VIII
Um Estudo Comparativo entre soluções MVC e WebFlux

Discente: Daniel Sehn Colao
Matrícula: 20230000401

Maio 2023

1 Introdução

Ao buscar soluções eficientes para problemas em diversas áreas, é comum encontrar duas abordagens distintas: reativa e interativa. Essas soluções diferem com base na velocidade de interação, sendo que as soluções reativas se concentram em reagir rapidamente a estímulos do ambiente, enquanto as interativas envolvem uma interação mais associada à máquina do sistema e o usuário.

A arquitetura reativa visa fornecer alta responsividade ao usuário, por meio das propriedades de resiliência e elasticidade, obtidas pela comunicação assíncrona e não bloqueante. Este modelo de comunicação aumenta a capacidade responsiva do sistema em razão de haver bloqueio de threads durante o atendimento da requisição ou comunicação a serviços externos. Em contrapartida, sistemas interativos trabalham, majoritariamente, com o modelo síncrono de comunicação, caracterizado por bloquear a execução da thread até a obtenção de resposta I/O, não sendo favorável às duas propriedades mencionadas.

Este trabalho tem como objetivo realizar uma comparação entre versões interativa (MVC) e reativa (WebFlux) de um sistema para loja virtual, ambas desenvolvidas na plataforma Spring. O desenvolvimento foi realizado com os módulos Spring MVC e Spring WebFlux, respectivamente.

O Project Loom se trata de uma iniciativa da Oracle que oferece o uso de *Lightweight Threads*, denominadas Virtual Threads, para o desenvolvimento de programas em Java. Esse projeto visa desacoplar threads da JVM do Sistema Operacional, proporcionando maior desempenho no gerenciamento de threads. Neste trabalho, também será analisado o impacto dessa abordagem nas métricas de desempenho.

2 Descrição da Aplicação

O sistema deste trabalho é voltado para lojas de comércio virtual. Sua composição é de três microsserviços:

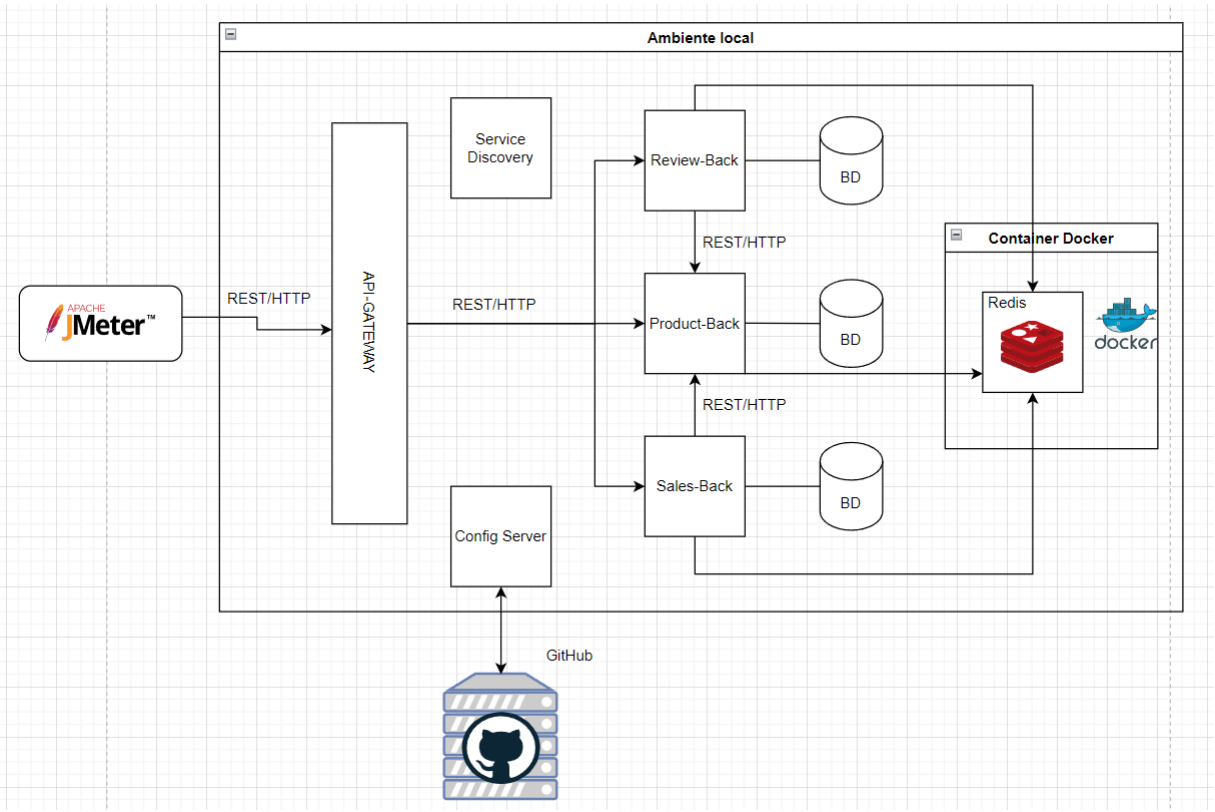
- **Product-back**: responsável pelo gerenciamento de produtos.
- **Review-back**: responsável pelo gerenciamento de avaliações de produtos.
- **Sales-back**: responsável por realizar vendas de produtos.

3 Arquitetura da aplicação

3.1 Componentes

- **Config Server**: Servidor de configuração. Local centralizado onde microsserviços obtêm suas configurações. Todas configurações estão armazenadas em um repositório remoto no GitHub.
- **Service Discovery**: Responsável por armazenar dinamicamente o endereço, porta e nome de cada microsserviço da aplicação.
- **API-Gateway**: Sua função principal é atuar como um ponto de entrada para todas as chamadas de API externas.
- **Redis**: Cache distribuído para agilizar a obtenção de dados, reduzindo o número de acessos ao banco de dados. Disponibilizado em um container Docker. O padrão Cache-Aside foi escolhido para o desenvolvimento deste trabalho.
- **BD**: Banco de dados de cada microsserviço. Sua função é persistir os dados da aplicação.
- **JMeter**: Software utilizado para realizar testes de carga no sistema desenvolvido.

3.2 Diagrama



4 Resultados - 1ª Entrega

Os resultados foram coletados a partir de testes de carga submetidos à rota de salvar uma avaliação acerca de um produto. URL: IP:porta/product-review. Método HTTP: POST.

O JMeter realizou requisições ao microsserviço *Review-back*, e este se comunicou com o microsserviço *Product-back* para verificar a existência do produto com base em seu nome. Foi necessário cadastrar, a priori, um produto para possibilitar o cadastro de avaliações.

O desempenho da aplicação foi medido com base na métrica *throughput*, o número de requisições atendidas por segundo. Os gráficos desta seção foram construídos conforme os resultados presentes na figura 1, encontrado na seção 4.1 deste relatório.

4.1 Tabela com resultados

Análise de Desempenho: Throughput (requisições/s)					
Aplicação	20 usuários	50 usuários	100 usuários	400 usuários	1000 usuários
MVC	41.6	100.5	196.7	167.8	84.3
MVC + Loom	41.6	100.8	196.9	155.2	80.3
WebFlux	40	95.3	182.4	318.3	320.6
WebFlux + Loom	41.2	97.9	196.4	361.7	513.5

Figura 1: Tabela com o *Throughput* de cada teste de carga feito com o JMeter.

4.2 Gráfico de Barras

O gráfico de barras da Figura 2 descreve o *Throughput*, medido em requisições atendidas por segundo, de cada teste de carga feito em cada versão da aplicação.

Neste trabalho, todas versões apresentaram comportamento similar até 100 usuários. A versão MVC foi superior ao WebFlux com 400 usuários simultâneos. No entanto, é notória a discrepância de desempenho a partir de 400 usuários, obtendo, aproximadamente, o dobro de desempenho nesse caso de teste.

Além disso, a solução reativa apresentou um comportamento mais estável, uma vez que o throughput não foi reduzido durante os testes, ao contrário da solução interativa, na qual o aumento dessa métrica foi interrompido depois de 200 usuários.

Com relação ao modelo de threads, ficou evidente sua contribuição a performance da aplicação apenas na a versão reativa (WebFlux) a partir de 400 usuários. Na versão interativa (MVC), não houve melhorias significativas.

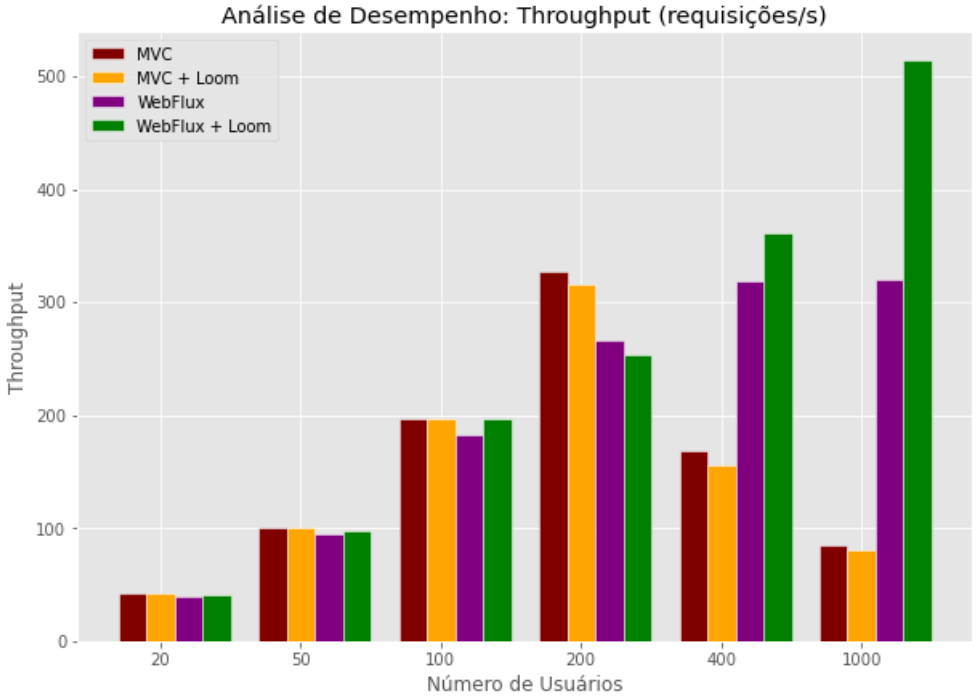


Figura 2: Gráfico de barras com os todos resultados obtidos nos testes de carga com o JMeter.

4.3 Gráfico de linhas - Versão Interativa (MVC)

O gráfico da Figura 3 exibe a diferença de desempenho da aplicação MVC com threads convencionais do Java, cujo mapeamento com threads do sistema operacional é 1-para-1, e a mesma aplicação porém com Virtual Threads do Project Loom.

O desempenho tornou-se desigual a partir de 100 usuários consumindo a aplicação simultaneamente. A vantagem do MVC em relação ao MVC + Loom foi relativamente pequena, apenas com 400 usuários que houve uma diferença de 10 requisições atendidas por segundo.

A solução não conseguiu manter throughput de maneira crescente ao longo dos testes, evidenciando um problema de escalabilidade em razão da abordagem de comunicação síncrona e bloqueante.

Além disso, a capacidade de joelho, *Knee Capacity*, é encontrada no momento de 100 usuários simultâneos, por ser o ponto no qual o throughtput reduz seu ritmo de crescimento.

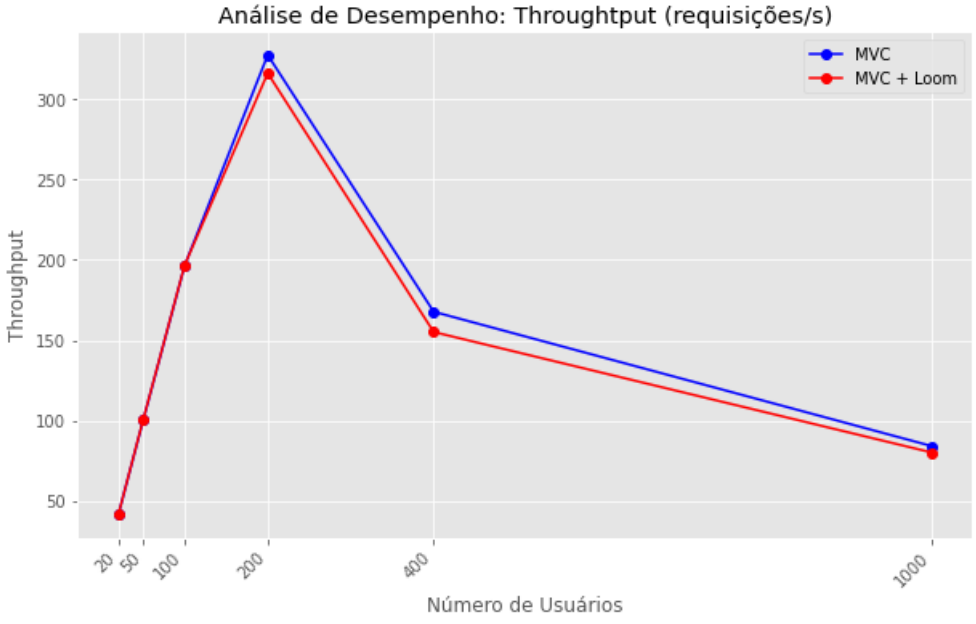


Figura 3: Gráfico de linhas com resultados obtidos no teste de carga com o JMeter para a aplicação MVC com e sem o uso de Virtual Threads.

4.4 Gráfico de linhas - Versão Reativa (WebFlux)

O desempenho tornou-se desigual a partir de 100 usuários simultâneos. A vantagem do WebFlux sobre o WebFlux + Loom foi relativamente pequena até esse ponto. O WebFlux + Loom abriu larga vantagem a partir de 400 usuários simultâneos, favorecendo a responsividade do sistema.

A solução conseguiu manter o crescimento do throughput ao longo dos testes, em razão da metodologia de comunicação e processamento síncronos e não bloqueante, em oposição ao MVC.

Nesta versão, a capacidade de joelho é encontrada no momento de 100 usuários simultâneos, assim como na versão MVC.

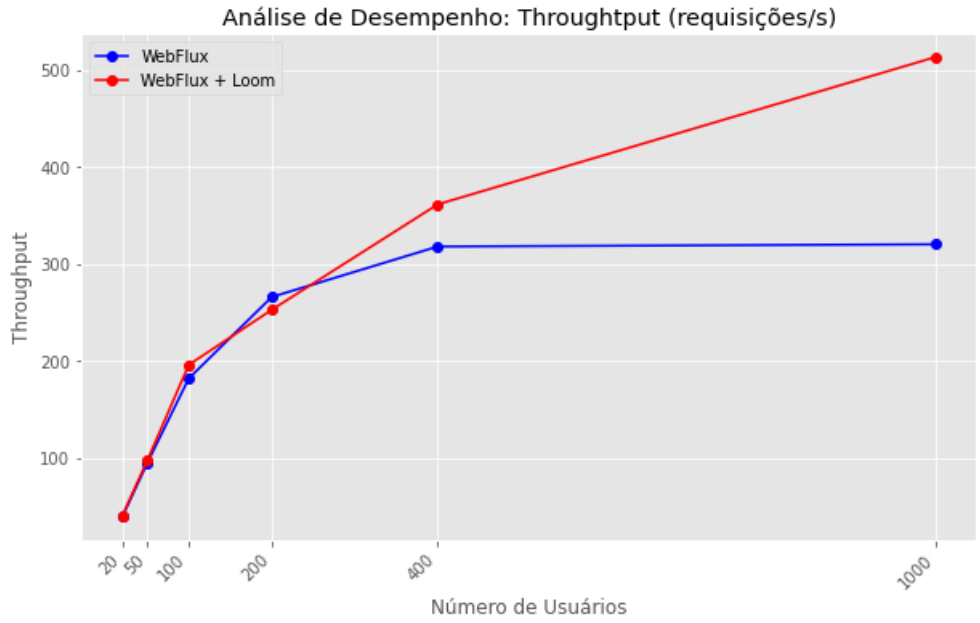


Figura 4: Gráfico de linhas com resultados obtidos no teste de carga com o JMeter para a aplicação WebFlux com e sem o uso de Virtual Threads.

4.5 Ambiente de Testes

O computador utilizado para os testes desse trabalho contém as seguintes especificações:

- Processador: Intel i7 8700 3GHz com 6 núcleos e 12 Threads.
- Memória RAM: 16GB (2x8GB) DDR4 2666MHz.
- SSD: 250GB.
- HDD: 1TB WD Blue.
- SO: Windows 10.
- Linguagem de programação: Java 19.

5 Resultados - 2ª Entrega

Na segunda etapa deste trabalho, foi empregado o componente Cache para analisar o impacto ocasionado na performance da aplicação.

Os resultados foram coletados a partir de testes de carga submetidos as seguintes rotas do módulo *Product-Back*:

- Cadastrar produto (POST).
- Buscar produto por ID (GET).
- Buscar todos produtos (GET).

5.1 Tabela com resultados

Análise de Desempenho: Throughput (requisições/s)							
Aplicação\Nº de usuários	20 usuários	50 usuários	100 usuários	200 usuários	400 usuários	1000 usuários	Tx de acerto médio (cache)
MVC	440 req/s	1140 req/s	1910 req/s	2000 req/s	2070 req/s	1800 req/s	~
MVC + Cache	450 req/s	1100 req/s	1900 req/s	2100 req/s	2200 req/s	2230 req/s	80%
WebFlux	472 req/s	1140 req/s	2243 req/s	2440 req/s	2600 req/s	2660 req/s	~
WebFlux + Cache Distribuído	467 req/s	1130 req/s	2100 req/s	2300 req/s	2500 req/s	2650 req/s	34%
WebFlux + Cache Local	472 req/s	1100 req/s	2241 req/s	2300 req/s	2700 req/s	2900 req/s	BUGGED

Figura 5: Tabela com o *Throughput* de cada teste de carga feito com o JMeter.

5.2 Gráfico de Barras

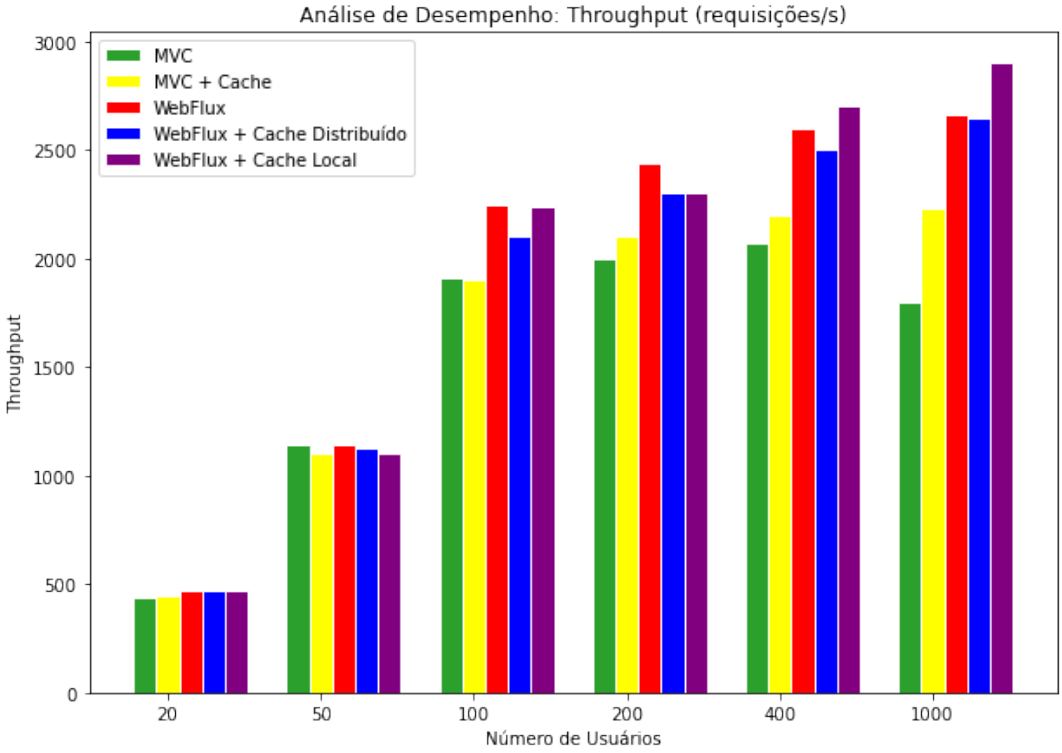


Figura 6: Gráfico de barras com os resultados das cinco versões analisadas.

5.3 Gráfico de Linhas - Versão MVC

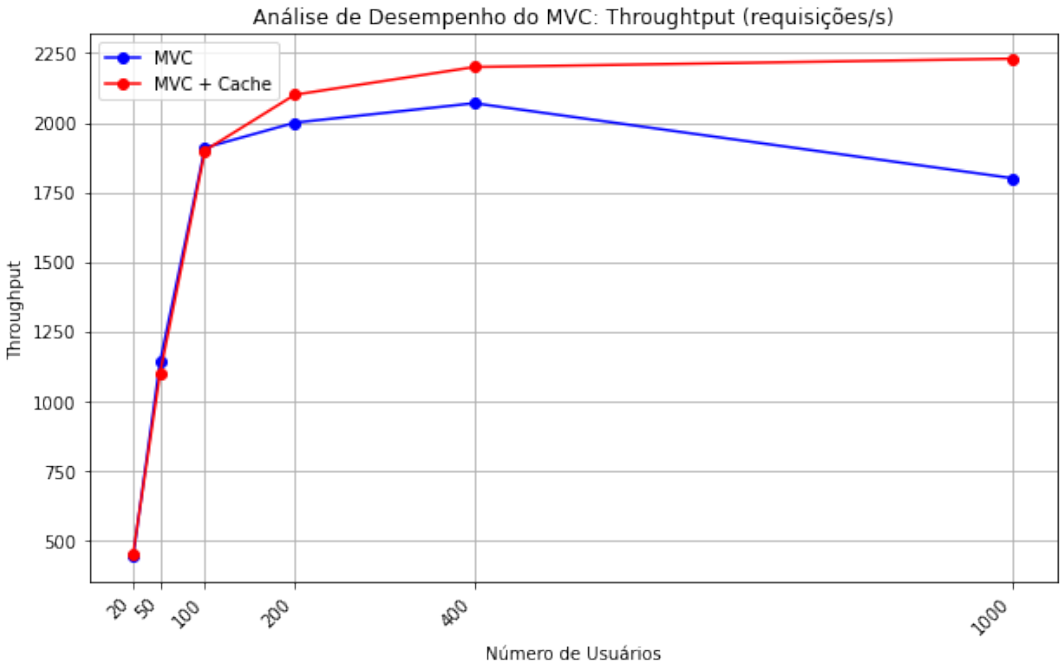


Figura 7: Gráfico de linhas com os resultados das duas versões do MVC.

5.4 Gráfico de Linhas - Versão Reativa (WebFlux)

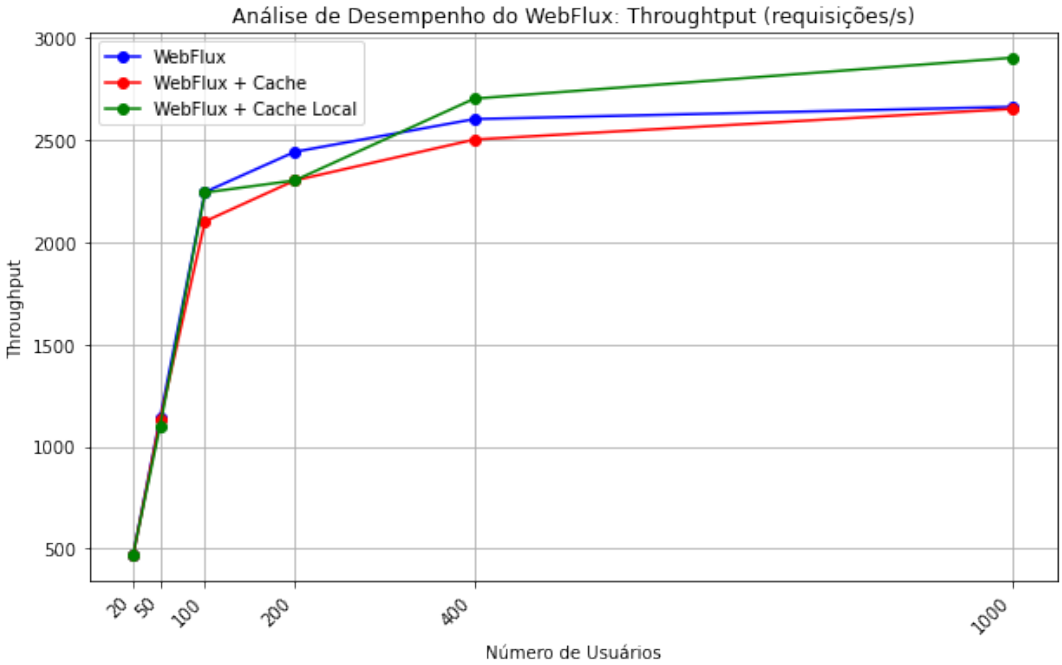


Figura 8: Gráfico de linhas com os resultados das três versões do WebFlux.

5.5 Ambiente de Testes

O computador utilizado para os testes desta 2ª etapa do trabalho contém as seguintes especificações:

- Processador: Intel i7 8700 3GHz com 6 núcleos e 12 Threads.
- Memória RAM: 16GB (2x8GB) DDR4 2666MHz.
- SSD: 250GB.
- HDD: 1TB WD Blue.

- SO: Windows 10.
- Linguagem de programação: Java 19.

6 Conclusão

Com base no que foi apresentado neste relatório, conclui-se que aplicações reativas tendem a desempenhar melhor que interativas em momentos nos quais o número de acessos simultâneos é elevado, por causa de sua abordagem de comunicação e processamento assíncronos e não bloqueante, favorecendo a propriedade de escalabilidade, fornecendo maior responsividade ao usuário.