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.
  Repositório (MVC): https://github.com/1DanielSC/product-back
- **Review-back**: responsável pelo gerenciamento de avaliações de produtos.
  Repositório (MVC): https://github.com/1DanielSC/review-back
- **Sales-back**: responsável por realizar vendas de produtos.
  Repositório (MVC): https://github.com/1DanielSC/sales-back
  As versões reativas se encontram no repositório https://github.com/1DanielSC/reactive-architecture

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.
  Link repositório: https://github.com/1DanielSC/Config-Server
  Link repositório com configurações de cada aplicação: https://github.com/1DanielSC/git-config-repo
- **Service Discovery**: Responsável por localizar e obter informações sobre os serviços disponíveis.
  Link repositório: https://github.com/1DanielSC/eureka-server

- **API-Gateway**: Sua função principal é atuar como um ponto de entrada para todas as chamadas de API externas.

  Link repositório: https://github.com/1DanielSC/gateway-server

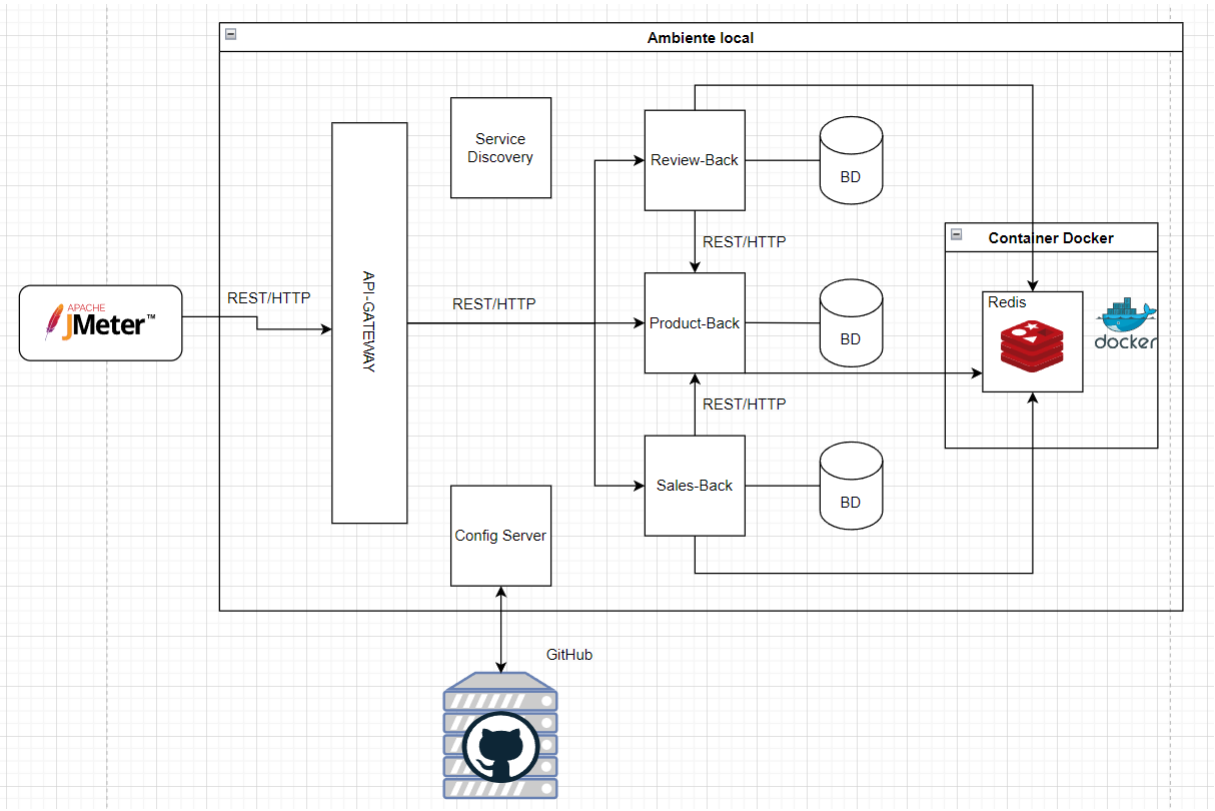
- **Redis**: Cache distribuído para agilizar a obtenção de dados, reduzindo o número de acessos ao banco de dados. Disponibilizado em um container Docker. O padrão Cache-Aside foi escolhido para o desenvolvimento deste trabalho.

- **BD**: Banco de dados de cada microsserviço. Sua função é persistir os dados da aplicação.

  Foi utilizado banco de dados MongoDB para execução dos testes.

- **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.

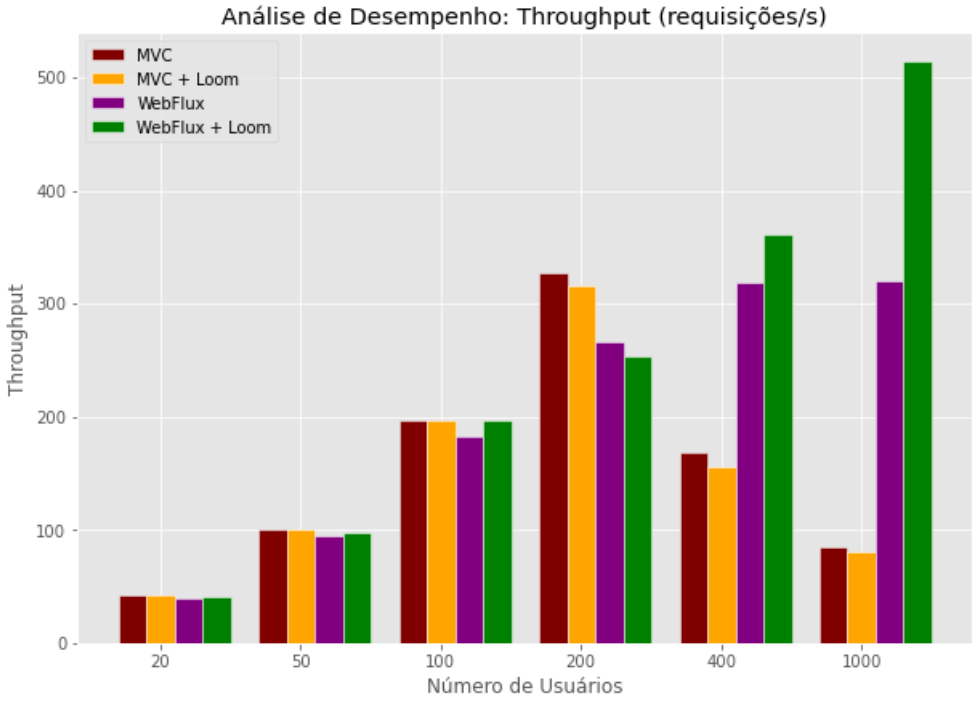


Figura 2: Gráfico de barras com os todos resultados obtidos nos testes de carga com o JMeter.

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.

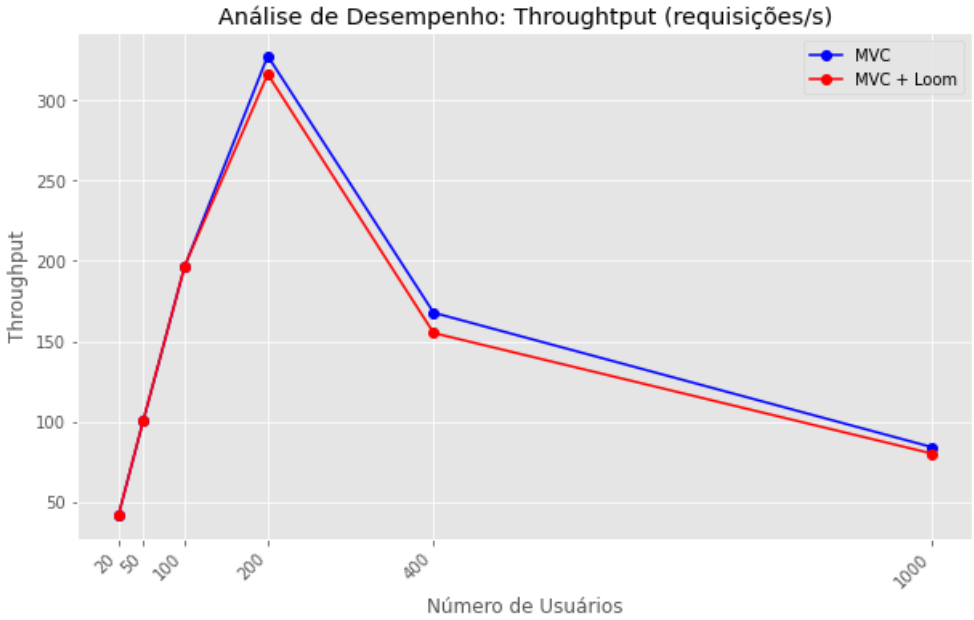


Figura 3: Gráfico de linhas com resultados obtidos no teste de carga com o JMeter para a aplicação MVC com e sem o uso de Virtual Threads.

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.

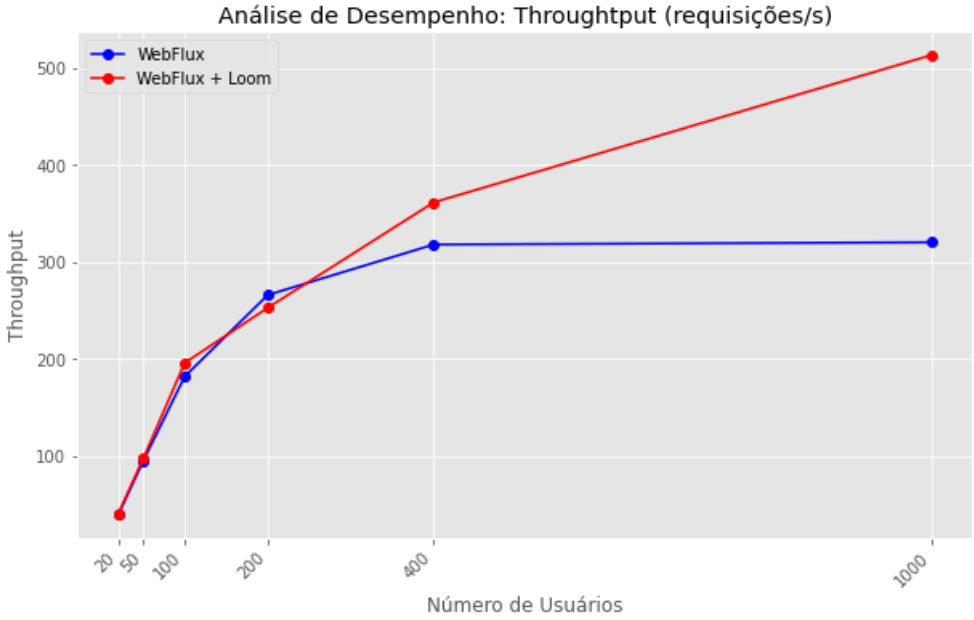


Figura 4: Gráfico de linhas com resultados obtidos no teste de carga com o JMeter para a aplicação WebFlux com e sem o uso de Virtual Threads.

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

A tabela descrita na Figura 5 contém todas métricas coletadas a partir dos testes de carga realizados na ferrramenta JMeter. O throughput foi calculado com base no número de requisições atendidas por segundo.

A coluna "Tx de acerto médio"indica a métrica calculada a partir dos valores cache hit e cache miss no servidor Redis.

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	0 hits ; 2 misses = 0%

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

Em todas versões, a taxa de erro permaneceu em 0%. Dessa forma, foi ocultada nessa tabela por questões de espaço.

Como duas versões foram testadas sem o Cache, linhas 1 e 3 da tabela acima respectivamente, a métrica Taxa de acerto médio não foi mensurada.

Além disso, a taxa de acerto médio obtida na versão WebFlux + Cache Local indica que o cache local foi utilizado com sucesso, em razão do serviço distribuído Redis ter sido solicitado apenas 2 vezes.

5.2 Gráfico de Barras

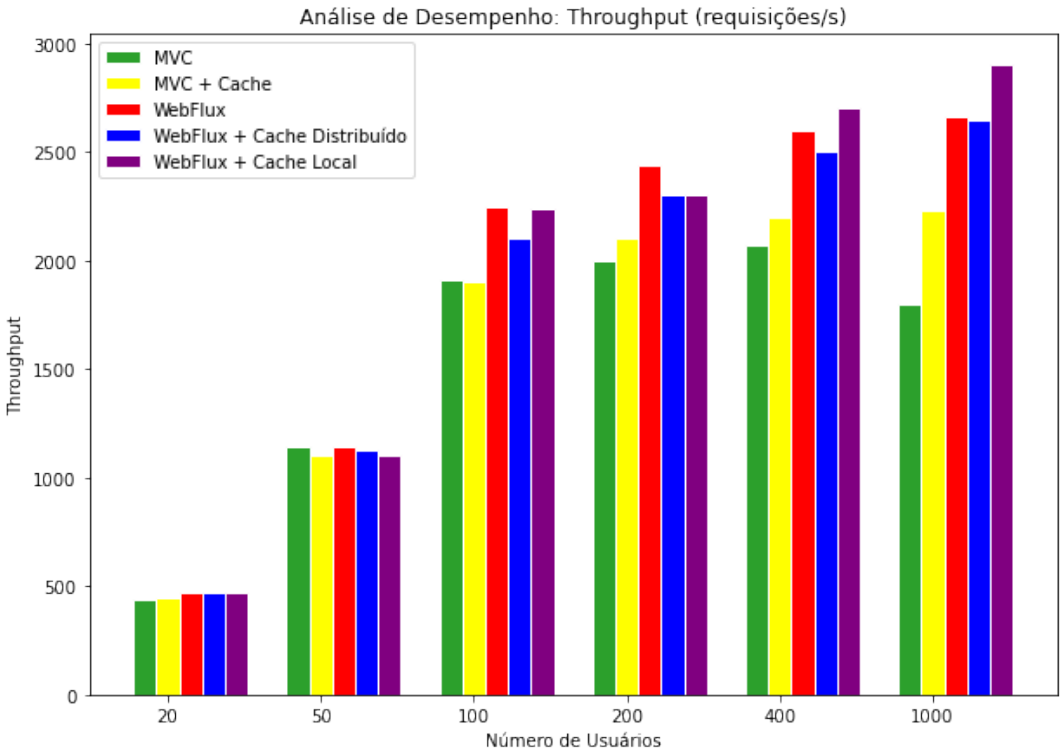


Figura 6: Gráfico de barras com os resultados das cinco versões analisadas.

5.3 Gráfico de Linhas - Versão MVC

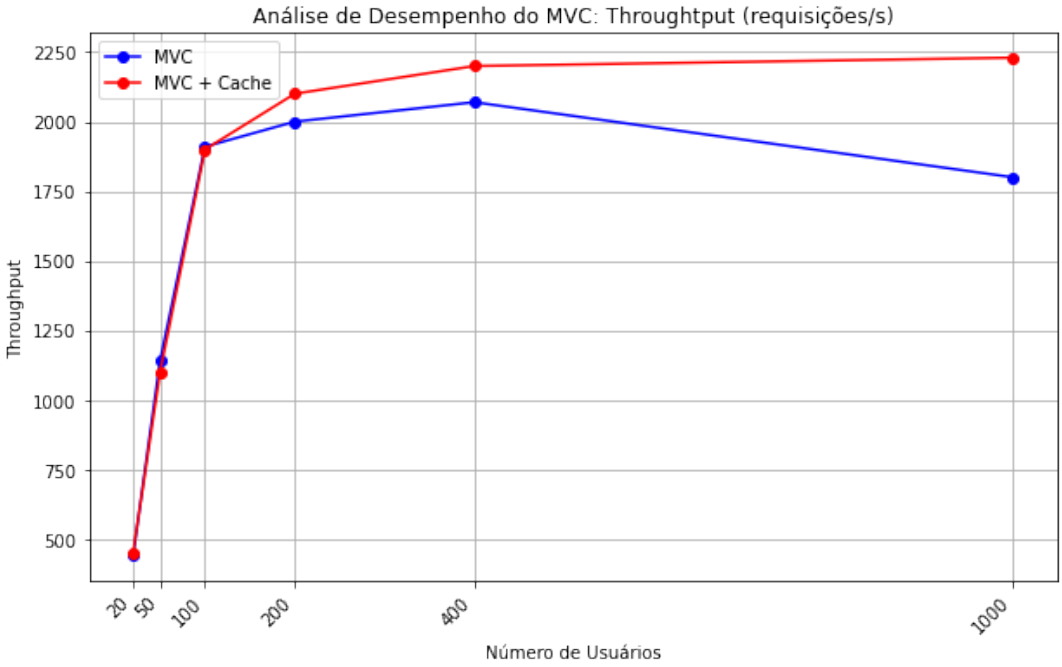


Figura 7: Gráfico de linhas com os resultados das duas versões do MVC.

A partir da Figura 7, observa-se que a versão MVC com cache não atingiu sua capacidade de uso no momento de 1000 usuários simultâneos, uma vez que seu throughput continuou a crescer.

A versão MVC sem cache atingiu sua capacidade de uso no momento de 400 usuários simultâneos, ponto crítico no qual a performance começa a decair.

Isso demonstra, nesse experimento, o aumento de desempenho que o uso de cache pode proporcionar às aplicações, ao evitar comunicação direta com o banco de dados.

5.4 Gráfico de Linhas - Versão Reativa (WebFlux)

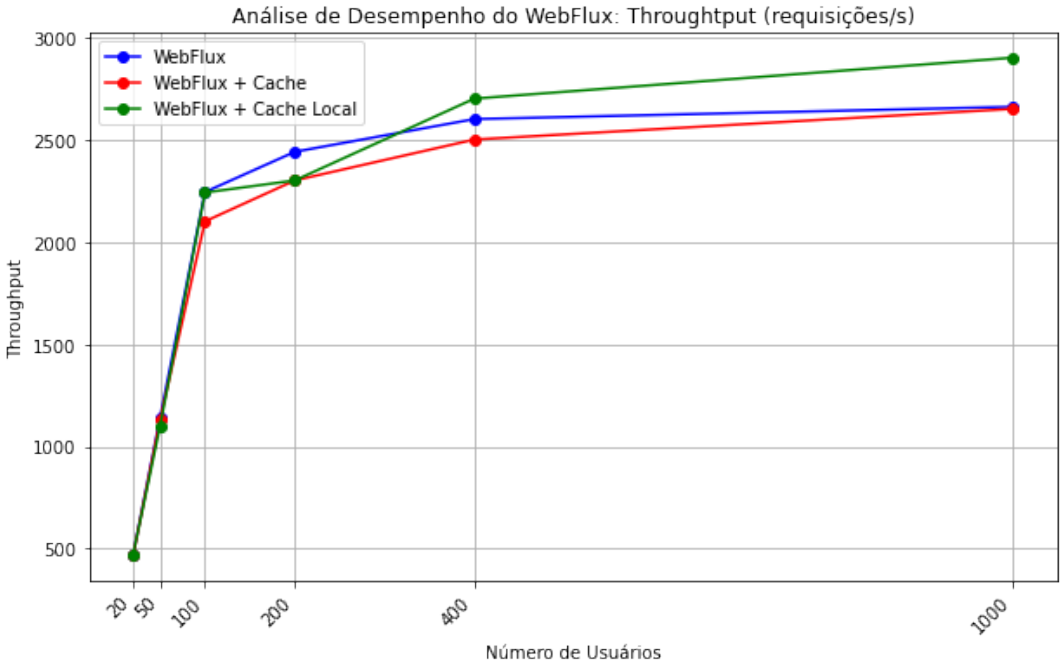


Figura 8: Gráfico de linhas com os resultados das três versões do WebFlux.

Com base no gráfico de linhas (Figura 8), é possível observar que as três versões reativas apresentaram resultados semelhantes até atingirem cerca de 400 usuários simultâneos. Além disso, a capacidade de joelho (**Knee Capacity**) foi atingida em torno de 100 usuários pelas três versões.

A versão com cache local se distanciou das demais versões a partir de 400 usuários, atingindo quase 3000 requisições/s com 1000 usuários. A versão webflux apresentou uma lateralização no throughput, uma vez que esta métrica tornou-se constante a partir de 400 usuários até 1000 usuários, enquanto a versão webflux com cache distribuído ainda conseguiu aumentar a vazão de requisições nesse intervalo.

Apenas a versão WebFlux sem cache apresentou decaimento de perfomance a partir de 400 usuários simultâneos.

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 Arquitetura e Resultados - 3ª Entrega

A terceira etapa deste trabalho caracteriza-se pela mudança de arquitetura implantada até a etapa anterior. A nova versão consiste em migrar para a arquitetura orientada a eventos, a qual emprega a comunicação assíncrona para transmissão de eventos e/ou mensagens. Essa forma de comunicação desempenha um papel crucial para permitir o processamento eficiente de eventos e a troca de mensagens entre componentes do sistema, tornando-o mais escalável. Como um dos pilares do Manifesto Reativo, ela contribui para a construção de sistemas resilientes e responsivos.

Foram implementados dois padrões de projeto, baseados em comunicação assíncrona, para a migração de arquitetura. Os padrões implementados foram:

- Event Notification

  No padrão de notificação de eventos, um componente envia uma mensagem ou evento para notificar outros componentes sobre uma determinada ação, mudança de estado ou evento significativo. Esse padrão foi utilizado em conjunto com o Request-Reply.

- Request-Reply

  Trata-se de um padrão de troca de mensagens no qual um requisitor envia uma mensagem de solicitação para um serviço replicador, que recebe e processa a solicitação, retornando uma mensagem em resposta.

  São utilizadas duas filas ou logs, a depender do broker empregado, para realizar a troca de mensagens entre os dois serviços.

  Este padrão foi aplicado na comunicação Order-Back e Product-Back para o caso de verificar se um produto possui quantidade suficiente para uma determinada compra, assim como Review-Back e Product-Back para verificar se a avaliação a ser cadastrada é sobre um produto existente no sistema.

6.1 Arquitetura da Aplicação

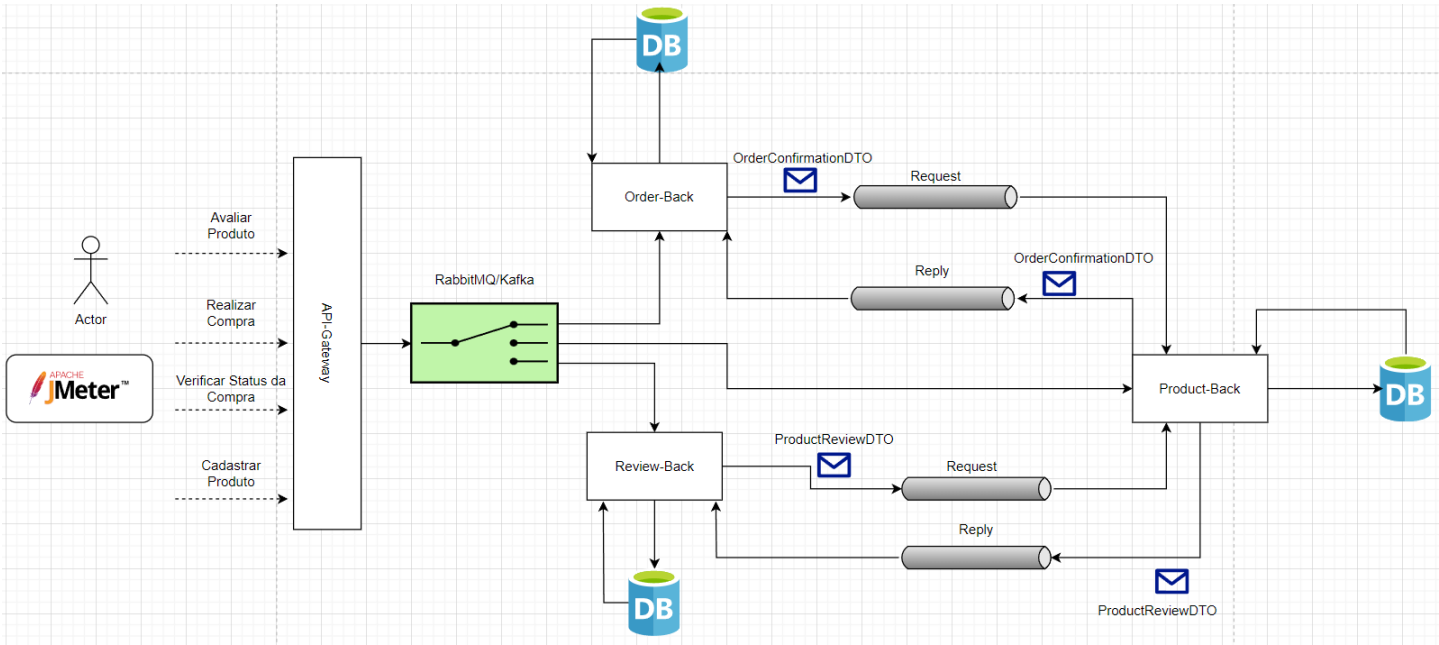


Figura 8: Arquitetura da aplicação orientada a eventos.

6.2 Resultados

A rota de criar uma avaliação de produto foi utilizada para avaliar o desempenho da versão orientada a eventos. Os resultados obtidoas na primeira etapa do trabalho, versões MVC e WebFlux, foram reaproveitados com o objetivo de comparar com a nova arquitetura.

6.2.1 Gráfico de Barras

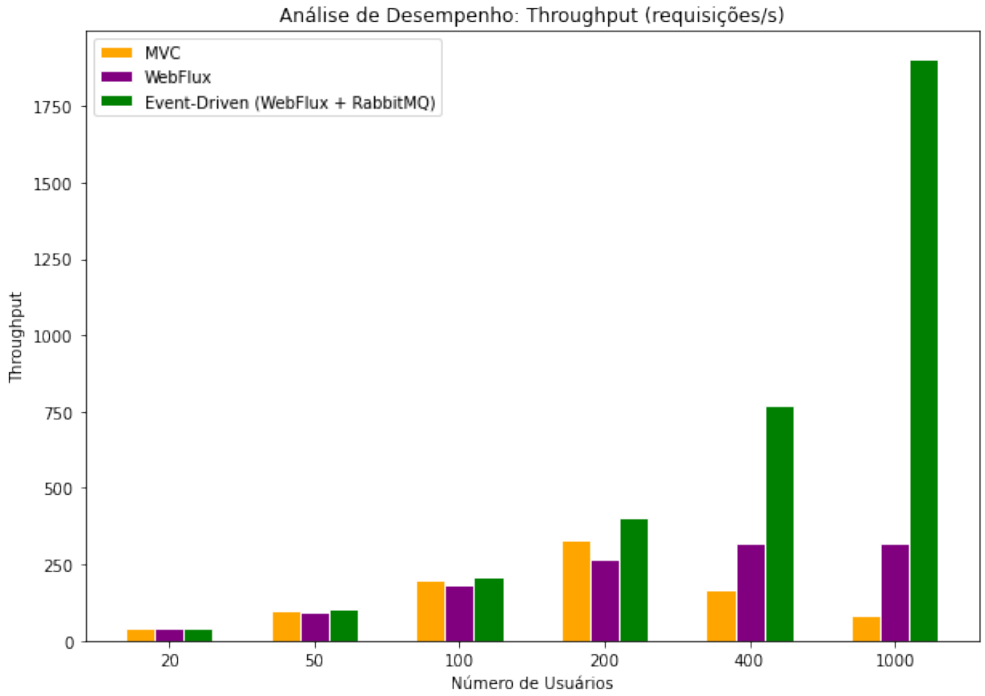


Figura 9: Gráfico de barras com resultados das versões MVC, WebFlux e Event-Driven.

6.2.2 Gráfico de Linhas

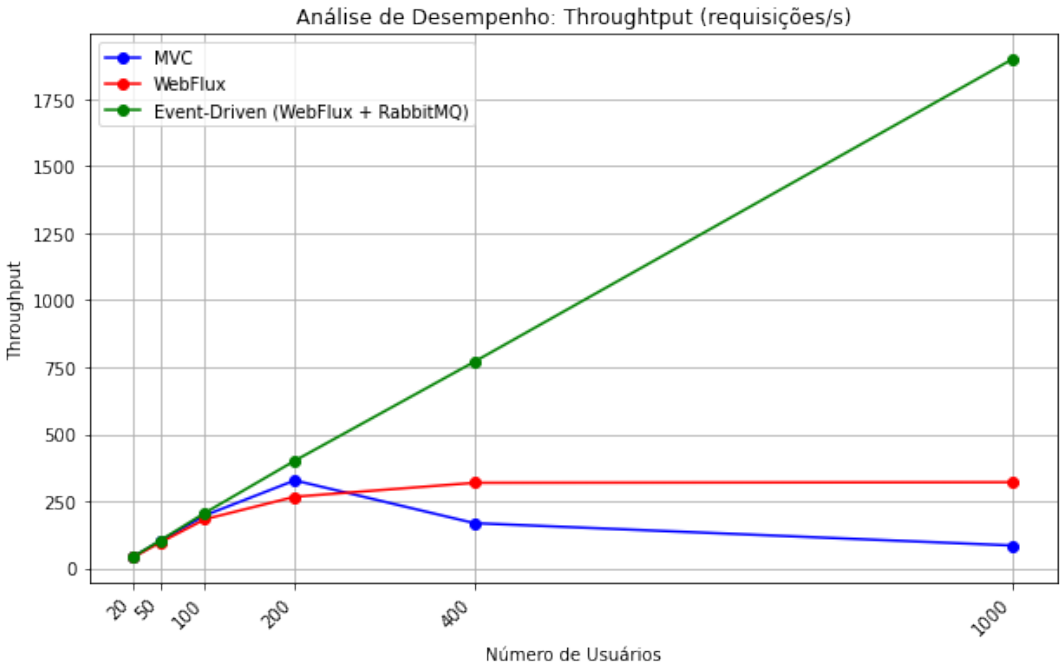


Figura 10: Gráfico de linhas com resultados das versões MVC, WebFlux e Event-Driven.

7 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, a qual fornece maior responsividade ao usuário.

Ademais, o uso de Cache favoreceu o ganho de perfomance nas aplicações desenvolvidas, aumentando a vazão de requisições atendidas por segundo. Isso resulta em uma maior escalabilidade da aplicação, permitindo lidar com um aumento de carga sem comprometer o desempenho do sistema.