Impacto de Desempenho da Granularidade de Microsserviços: Uma Avaliação com o Arcabouço Service Weaver

Ricardo César Mendonça Filho[1], **Nabor C. Mendonça**[2]

[1]Centro de Ciências Tecnológicas
[2]Programa de Pós-Graduação em Informática Aplicada
Universidade de Fortaleza (UNIFOR)
Fortaleza – CE

{ricardocesarfilho,nabor}@unifor.br

Abstract. *The Service Weaver framework enables the development of distributed applications in Go as modular monoliths, with the flexibility to deploy monolith components in different environments and at various levels of granularity without code changes. This work evaluates the performance of an open-source microservice application using Service Weaver, considering multiple service granularities in environments of one and two virtual machines under various workloads. The results indicate that service decoupling, while beneficial to modularity and maintenance, can significantly increase communication overhead between processes and virtual machines, negatively affecting the application's performance and scalability. These findings highlight the importance of balancing service granularity and communication costs in the design and deployment of microservices applications.*

Resumo. *O arcabouço Service Weaver permite desenvolver aplicações distribuídas em Go como monólitos modulares, com flexibilidade para implantar os componentes do monólito em diferentes ambientes e níveis de granularidade sem alterações de código. Este trabalho avalia o desempenho de uma aplicação de microsserviços de código aberto com o Service Weaver, considerando múltiplas granularidades de serviço em ambientes de uma e duas máquinas virtuais, sob variadas cargas de trabalho. Os resultados indicam que o desacoplamento de serviços, embora beneficie a modularidade e a manutenção, pode aumentar significativamente a sobrecarga de comunicação entre processos e máquinas virtuais, afetando negativamente o desempenho e escalabilidade da aplicação. Esses achados destacam a importância de balancear a granularidade dos serviços e os custos de comunicação no projeto e implantação de aplicações de microsserviços.*

1. Introdução

Um dos desafios cruciais no projeto de aplicações baseadas em microsserviços é determinar o nível de granularidade ideal para cada serviço [Jamshidi et al. 2018, Vera-Rivera et al. 2021]. Se os serviços são grandes demais, perdem-se os benefícios do acoplamento fraco e da independência. Se são pequenos demais, pode haver complexidade desnecessária e sobrecarga de comunicação, afetando o desempenho e a escalabilidade da aplicação [Newman 2019]. Soluções tradicionalmente utilizadas

no desenvolvimento de microsserviços não são adequadas para enfrentar esses desafios [Ghemawat et al. 2023]. A principal razão é que todas presumem que o desenvolvedor divida manualmente sua aplicação em vários serviços independentes. Isso significa que a topologia de comunicação da aplicação é predeterminada pelo desenvolvedor da aplicação. Na prática, essa restrição reduz as opções de granularidade que podem ser exploradas pelos desenvolvedores de microsserviços, forçando-os a tomar decisões de projeto sub-ótimas prematuramente [Dragoni et al. 2017].

O arcabouço Service Weaver, lançado pela Google no início de 2023, é uma solução de código aberto com potencial para endereçar os desafios de granularidade discutidos acima [Google 2024]. Com o Service Weaver, os desenvolvedores podem escrever suas aplicações distribuídas em Go como um monólito modular. Posteriormente, em tempo de implantação, o arcabouço divide esse monólito em múltiplos serviços distribuídos, de acordo a topologia de comunicação definida pelo desenvolvedor, sem necessidade de mudanças no código do monólito. Isso permite que os serviços da aplicação sejam facilmente implantados localmente ou na nuvem, sob diferentes topologias de comunicação, com um mínimo de esforço de configuração [Ghemawat et al. 2023]. Nesse sentido, o Service Weaver inova ao combinar as vantagens de um monólito modular com a flexibilidade e escalabilidade dos microsserviços [Campbell 2023].

Este trabalho tem como objetivo responder as seguintes questões de pesquisa:

RQ1 Qual é o impacto no desempenho de uma aplicação distribuída quando os serviços da aplicação são implantados com diferentes níveis de granularidade?

RQ2 Como o impacto no desempenho da aplicação é influenciado por fatores como a carga de trabalho esperada e a alocação dos serviços aos recursos da infraestrutura física subjacente?

Para responder essas duas questões, o trabalho: (i) selecionou uma aplicação distribuída de código aberto amplamente adotada como *benchmark* de pesquisa pela comunidade de microsserviços; e (ii) utilizou a alta flexibilidade de implantação de serviços originalmente introduzida pelo arcabouço Service Weaver para testar o desempenho da aplicação selecionada considerando diferentes configurações de implantação dos serviços, sob diferentes cargas de trabalho, em cenários com uma e duas máquinas virtuais (VMs).

Os resultados obtidos revelam que a escolha da granularidade dos serviços em arquiteturas de microsserviços tem um impacto significativo no desempenho, especialmente em relação às estratégias de comunicação entre processos localizados em uma mesma VM ou em diferentes VMs. As configurações testadas em uma única VM indicaram que, embora o desacoplamento dos serviços possa oferecer benefícios iniciais em termos de modularidade e manutenção, a sobrecarga na comunicação entre processos pode se tornar um fator limitante sob altas cargas de trabalho. Em contraste, as configurações distribuídas em duas VMs evidenciaram que a comunicação entre VMs introduz uma camada adicional de complexidade e latência, particularmente sob condições de carga elevada. Esses achados demonstram que a granularidade dos serviços e a distribuição destes entre vários processos e VMs são aspectos críticos a serem considerados no projeto e na implantação de aplicações de microsserviços, influenciando diretamente no seu desempenho e escalabilidade.

O restante deste trabalho está organizado da seguinte forma: a Seção 2 oferece uma visão geral do arcabouço Service Weaver; a Seção 3 apresenta o método de avaliação

experimental utilizado; a Seção 4 analisa e discute os resultados obtidos; a Seção 5 compara os resultados do trabalho com outros trabalhos relacionados; por fim, a Seção 6 oferece as considerações finais e sugere tópicos para trabalhos futuros.

2. Service Weaver

Service Weaver é um arcabouço desenvolvido para facilitar a escrita, implantação, e gerenciamento de aplicações distribuídas na linguagem Go [Google 2024]. As duas principais inovações introduzidas pelo arcabouço são: (i) um modelo de programação que permite desenvolvedores escreverem aplicações distribuídas modulares, focadas apenas em regras de negócio, compiladas para um único artefato binário, e (ii) um ambiente de execução para a distribuição e gerenciamento otimizado dessas aplicações [Ghemawat et al. 2023].

O modelo de programação proposto possibilita que o programador escreva uma aplicação distribuída como um único programa, onde o código é dividido por unidades modulares, chamadas *componentes*. O arcabouço desacopla a implementação de regras de negócio da configuração e gerenciamento de recursos de infraestrutura: os componentes são centrados em torno de fronteiras lógicas baseadas na regra de negócios da aplicação, e o ambiente de execução do arcabouço é centrado em torno de fronteiras físicas baseadas no desempenho da aplicação (por exemplo, dois componentes podem ser co-localizados para melhorar o desempenho). Esse desacoplamento, juntamente com o fato de que as fronteiras físicas podem ser alteradas dinamicamente, contribui para aliviar os desafios de modularidade e granularidade causados por APIs arquiteturalmente rígidas, comuns nas arquiteturas distribuídas modernas [Ghemawat et al. 2023].

Em tempo de implantação, o ambiente de execução do Service Weaver analisa o código fonte da aplicação, gerando o código necessário para lidar com as chamadas remotas entre os serviços, o que permite que sejam executados em processos ou máquinas diferentes, sem que haja necessidade da implementação explícita de chamadas remotas pelo desenvolvedor. Além disso, o arcabouço administra o escalonamento, replicação e co-locação dos serviços criados, sem que seja necessária intervenção do desenvolvedor [Google 2024].

2.1. Componentes

Uma aplicação Service Weaver é composta por um conjunto de componentes. Um componente é descrito como uma interface implementada em Go, e componentes podem comunicar-se entre si por meio de chamadas de métodos definidos nestas interfaces. Esta característica permite a abstração de mecanismos de chamadas remotas e serialização de dados, os quais não precisam ser explicitamente implementadas pelo desenvolvedor da aplicação [Google 2024].

Os componentes podem ser hospedados em diferentes processos do sistema operacional e possivelmente em múltiplas máquinas. As invocações de métodos de componentes se transformam em chamadas de procedimentos remotas quando necessário, mas permanecem como chamadas de procedimento local se o componente chamador e o componente chamado estiverem no mesmo processo [Ghemawat et al. 2023].

A interface Go de um componente estabelece os métodos do componente e a anotação *weaver.Implements* declara a interface como um componente do arcabouço. A

```go
// Definition of an Adder component with an Add method
type Adder interface {
    Add(ctx context.Context, x, y int) (int, error)
}
type adder struct {
    weaver.Implements[Adder]
}
func (a adder) Add(ctx context.Context, x, y int) (int, error) {
    return x + y, nil
}
```

Listagem 1: Exemplo de declaração de um componente no Service Weaver.

```go
func main() {
    ctx := context.Background()
    root := weaver.Init(ctx)
    // Get and call the Adder component
    adder, err := weaver.Get[Adder](root)
    if err != nil {
        log.Fatal(err)
    }
    sum, _ := adder.Add(ctx, 1, 2)
    fmt.Printf("Sum: %d", sum)
}
```

Listagem 2: Exemplo de invocação do componente declarado na Listagem 1.

Listagem 1 mostra um exemplo da declaração de um componentes do Service Weaver em Go, de nome *Adder*. Esse componente implementa o método *Add*, nas linhas 8–10, que retorna a soma dos valores de dois parâmetros do tipo inteiro. O método desse componente pode ser invocado a partir de um outro componente no Service Weaver utilizando apenas chamadas locais, como ilustra a Listagem 2.

2.2. Geração de Código

A principal responsabilidade do ambiente de execução do Service Weaver envolve a geração de código da aplicação a ser implantada. Ao examinar as anotações do tipo *Implements[<type>]* no código-fonte de um programa, o gerador de código determina o conjunto coletivo de interfaces e implementações de componentes. Posteriormente, ele produz código para facilitar a conversão de argumentos a partir dos métodos do componente. Além disso, ele cria código para executar esses métodos como chamadas remotas, no caso do componente chamado ser executado em um processo diferente do processo do componente chamador. O código resultante é compilado junto com o código do desenvolvedor em um binário unificado [Ghemawat et al. 2023].

2.3. Configuração de Implantação

O Service Weaver permite que o desenvolvedor defina explicitamente aspectos de infraestrutura e comunicação da aplicação a serem considerados em tempo de implantação, como quais componentes devem ser executados de maneira acoplada, em um único processo, ou de maneira desacoplada, em múltiplos processos, possivelmente em múltiplas máquinas. A Listagem 3 mostra um exemplo de uma possível configuração de implantação dos componentes *main* e *Adder*, definida no formato TOML [Preston-Werner 2024]. Nesse exemplo, os dois componentes são implantados de forma acoplada, no mesmo processo (componentes não referenciados explicitamente no arquivo de configuração são implantados de

```
# weaver.toml file
[serviceweaver]
binary = "./app"
# Colocate the main and Adder components
colocate = [
    [
        "main",
        "Adder"
    ]
]
```

Listagem 3: Exemplo de uma configuração de implantação no Service Weaver.

forma desacoplada uns dos outros por padrão). Como explicado anteriormente, em tempo de execução, o Service Weaver realiza a comunicação entre os componentes acoplados via chamadas locais.

Embora simples, esse exemplo ilustra como é fácil um desenvolvedor configurar a topologia de comunicação de uma aplicação distribuída com o Service Weaver, efetivamente alterando a granularidade de implantação dos serviços da aplicação sem que seja necessário alterar o código fonte dos componentes de negócio. Essa facilidade dá mais flexibilidade aos desenvolvedores para experimentarem diferentes níveis de granularidade dos serviços, e avaliaram o impacto de desempenho e escalabilidade de cada um deles sob diferentes ambientes de implantação.

2.4. Modos de Implantação

O Service Weaver não inclui código específico para um ambiente de implantação particular. Em vez disso, o arcabouço utiliza uma API especial, chamada *Deployer* (implantador), para integrar a lógica da aplicação com os detalhes do ambiente de execução, como a distribuição física em máquinas ou regiões em uma nuvem pública. Isso permite que os desenvolvedores escrevam aplicações em Go independentes do ambiente, que podem ser implantadas em qualquer ambiente de execução compatível com os implantadores do Service Weaver. A versão mais recente do Service Weaver oferece vários tipos de implantadores que permitem implantar os componentes de uma aplicação em diferentes ambientes de execução, incluindo implantadores para Kubernetes e Google Cloud [Google 2024].

3. Método de Avaliação

Esta seção apresenta o método de avaliação adotado neste trabalho para responder às questões de pesquisa apresentadas na Seção 1, incluindo a aplicação de microsserviços selecionada, e o ambiente de teste e a ferramenta de geração de carga utilizados.

3.1. Aplicação Alvo

A aplicação alvo escolhida foi a Online Boutique [Google Cloud 2023]. Esta aplicação implementa uma loja virtual onde os usuários podem buscar produtos, adicionar produtos ao carrinho, e comprá-los. A aplicação foi originalmente desenvolvida pela Google, como forma de demonstrar um conjunto de tecnologias nativas de nuvem, como Kubernetes, GKE, Istio, e gRPC. Desde então, ela tem sido amplamente utilizada como *benchmark* de avaliação em diversos trabalhos de pesquisa envolvendo microsserviços (por exemplo, [Cui et al. 2020, Soldani et al. 2021, Park et al. 2021, Saleh Sedghpour et al. 2022]).

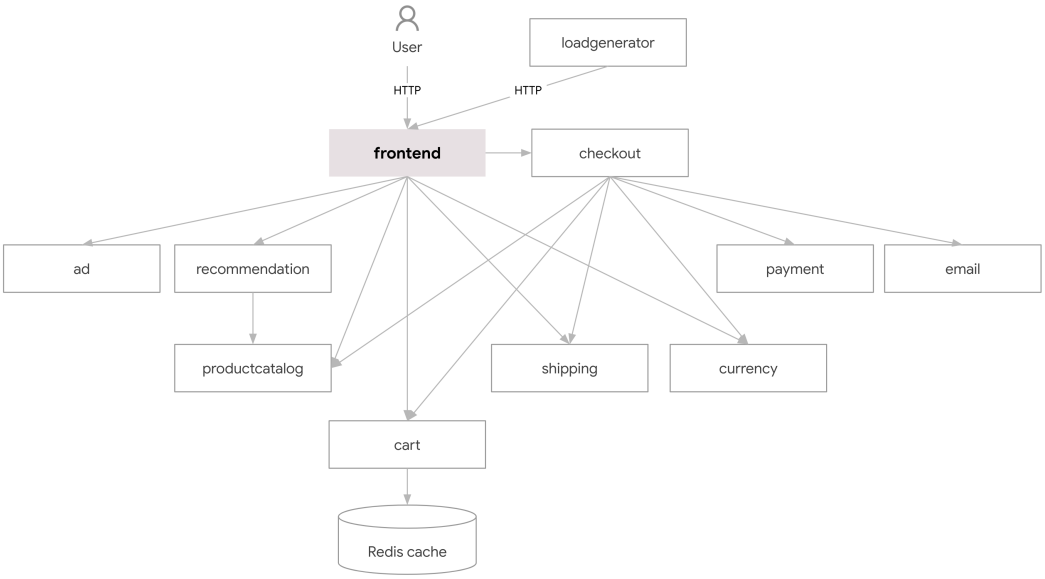


Figura 1. Arquitetura da aplicação Online Boutique (fonte: [Google Cloud 2023]).

A arquitetura da Online Boutique é composta por 11 microsserviços, responsáveis por diferentes aspectos do funcionamento da loja virtual, conforme ilustra a Figura 1. Na versão original da aplicação, esses serviços são implementados em diferentes linguagens de programação, como Go, Java, C#, Python e Node.js. Para que a aplicação pudesse ser executada utilizando o Service Weaver, os desenvolvedores do arcabouço portaram os demais serviços da Online Boutique para a linguagem Go, além de substituírem o uso do serviço de cache Redis por uma versão nativa em Go [Ghemawat et al. 2023]. Esta nova versão da aplicação escrita inteiramente em Go foi utilizada nos experimentos realizados neste trabalho.

3.2. Ambiente de Teste

Todos os teste foram conduzidos em uma única máquina hospedeira com as seguintes características: processador AMD Ryzen 7 5800H @ 3.2GHz, 8 núcleos de processamento, e 16GB de memória RAM DDR4. Durante os testes, um *cluster* do Kubernetes foi criado na máquina hospedeira utilizando a ferramenta Minikube [The Kubernetes Authors 2024]. Este *cluster* continha duas máquinas virtuais criadas com a ferramenta VirtualBox [Oracle 2024], cada uma contendo 4 CPUs e 4GB de memória RAM.

O processo de implantação da aplicação alvo nesse *cluster* foi realizado utilizando os arquivos de configuração gerados automaticamente pelo implantador do Kubernetes fornecido pelo Service Weaver. Esses arquivos são dinamicamente modificados para se adaptarem às arquiteturas desejadas, permitindo a criação de condições de infraestrutura específicas para cada cenário de teste. Essa flexibilidade na configuração dos arquivos de implantação permite ajustes precisos para simular diferentes ambientes e cenários, proporcionando um ambiente de testes robusto e adaptável.

A decisão de utilizar uma única máquina hospedeira com apenas duas máquinas virtuais como infraestrutura de teste foi motivada por três fatores. Primeiro, a aplicação alvo é relativamente pequena, com apenas 11 serviços não persistentes, o que torna

possível a implantação de todos os serviços em um único processo. Segundo, a utilização de uma única máquina hospedeira dedicada possibilita a realização de testes de desempenho de forma mais controlada, permitindo uma análise mais precisa do impacto de desempenho decorrente da granularidade de implantação dos serviços da aplicação. Por fim, a utilização de apenas duas máquinas virtuais simplifica a configuração e execução dos testes, reduzindo a complexidade e o tempo de execução dos experimentos.

Para facilitar a reprodução e possível extensão deste trabalho pela comunidade de pesquisa, todos os artefatos e dados de experimentação gerados estão disponíveis em `https://github.com/mendoncas/granularity-evaluation/`.

3.3. Geração de Carga e Métrica de Desempenho

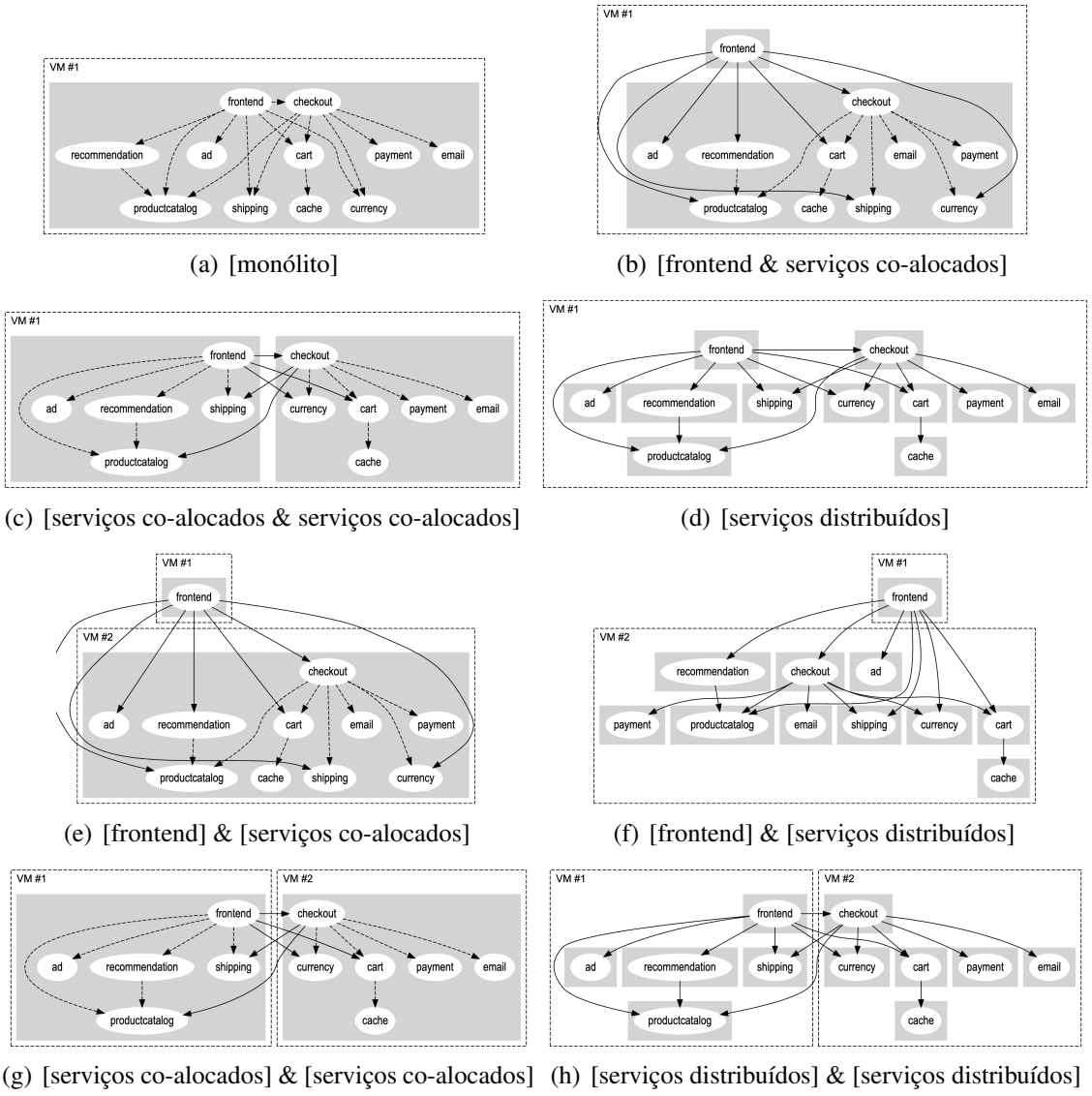
A ferramenta Locust [Locust.io 2024] foi utilizada para simular diferentes níveis de carga de trabalho sobre a aplicação alvo. Além de ser muito utilizada em testes de carga de aplicações web, a escolha desta ferramenta foi uma decisão natural, uma vez que a Online Boutique original utiliza o Locust como ferramenta de teste de carga e vem acompanhada de um *script* de teste criado especificamente para ela. Este *script* invoca uma série de operações oferecidas pelo serviço *frontend* da aplicação, simulando o comportamento de um típico usuário da loja:

- abrir a página inicial da loja;
- realizar duas trocas de moedas;
- realizar dez buscas por produtos;
- adicionar dois produtos ao carrinho de compras;
- realizar três visualizações do conteúdo do carrinho de compras; e
- efetuar o pagamento das compras (*checkout*).

Cada teste de carga envolveu a execução do Locust com o *script* acima por cerca de 5 minutos, com diferentes quantidades de usuários concorrentes, iniciando com 500 usuários até o limite de 2250 usuários, em incrementos de 250 usuários. Esses valores de carga foram definidos após a realização de um conjunto de testes iniciais, onde buscou-se determinar a quantidade máxima de usuários concorrentes suportada pela aplicação, quando implantada na infraestrutura física descrita anteriormente, sem a geração de respostas com erro.

Ao final de cada teste, o Locust produz um conjunto de métricas de desempenho coletadas durante o teste. Essas métricas incluem uma variedade de estatísticas calculadas sobre os tempos de resposta medidos para cada operação requisitada à aplicação alvo, como os tempos de resposta mínimo, máximo, médio, mediano, e diversos percentis, além da vazão e da taxa de erro de cada operação. O Locust também calcula estatísticas consolidadas, considerando o conjunto das operações executadas durante o teste de forma agregada.

Para os experimentos realizados neste trabalho, a métrica de desempenho utilizada foi o percentil 95 dos tempos de resposta, em milissegundos, calculado de forma agregada sobre o conjunto de operações requisitadas à aplicação por cada usuário. A escolha de um percentil elevado ($>$90%) como métrica para testes de desempenho é estrategicamente vantajosa, especialmente para software distribuído de larga escala, devido à sua capacidade de representar realisticamente o comportamento da aplicação em condições



(a) [monólito] (b) [frontend & serviços co-alocados]

(c) [serviços co-alocados & serviços co-alocados] (d) [serviços distribuídos]

(e) [frontend] & [serviços co-alocados] (f) [frontend] & [serviços distribuídos]

(g) [serviços co-alocados] & [serviços co-alocados] (h) [serviços distribuídos] & [serviços distribuídos]

Figura 2. Configurações de implantação da aplicação alvo em uma e duas VMs.

de estresse [Molyneaux 2014]. Ao contrário de métricas mais tradicionais, como média e mediana, métricas de percentil elevado evitam distorções causadas por valores atípicos, que tendem a acontecer com mais frequência em situações de sobrecarga.

3.4. Configurações de Implantação Avaliadas

Com o intuito de investigar como a granularidade de implantação de microsserviços pode afetar o desempenho de uma aplicação distribuída, os recursos do Service Weaver foram utilizados para definir e executar oito diferentes configurações de implantação dos onze microsserviços da aplicação Online Boutique, considerando cenários com uma e duas máquinas virtuais.

A Figura 2 ilustra essas oito configurações, sendo quatro em uma máquina virtual (Figuras 2(a)–2(d)) e quatro em duas máquinas virtuais (Figuras 2(e)–2(h)). Em todas as configurações, as máquinas virtuais são representadas por retângulos não preenchidos de borda tracejada, os processos, por retângulos preenchidos na cor cinza, e os componen-

tes da aplicação, por elipses na cor branca. As setas tracejadas representam chamadas locais, realizadas entre componentes executados em um mesmo processo, enquanto as setas sólidas representam chamadas remotas, realizadas entre componentes executados em processos diferentes.

Essas oito configurações foram definidas de modo a contemplar um conjunto reduzido mas representativo de um número potencialmente alto de possíveis configurações de implantação que poderiam ser utilizadas para alocar os onze microsserviços da aplicação alvo às duas VMs. As configurações escolhidas incluem de uma configuração monolítica (Figura 2(a)), executada em uma única máquina virtual, a configurações totalmente distribuídas (Figuras 2(f) e 2(h)), executadas em duas máquinas virtuais. Além disso, algumas configurações foram definidas de modo a isolar um serviço específico da aplicação dos demais serviços, como é o caso das configurações que isolam o serviço *frontend*, ilustradas nas Figuras 2(b), 2(e), e 2(f). Essas três configurações foram selecionadas dada a importância do serviço *frontend* como ponto de entrada da aplicação.

4. Resultados

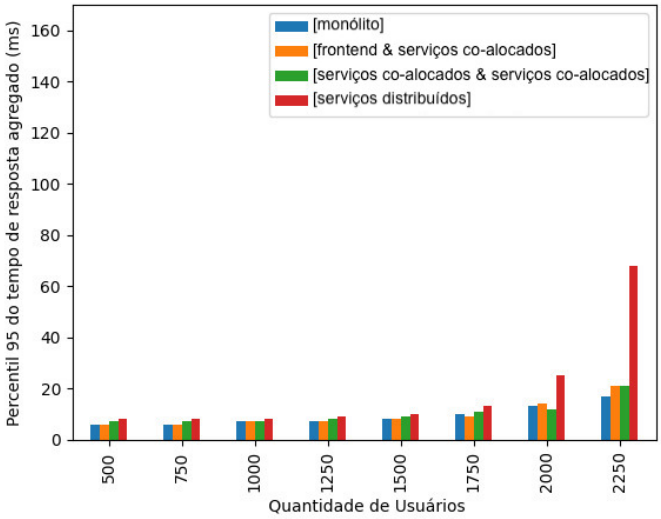
A Figura 3 mostra o desempenho das oito configurações de implantação da aplicação alvo, considerando cenários com uma e duas máquinas virtuais (VMs). Cada gráfico mostra o percentil 95 dos tempos de resposta agregados da aplicação medidos para cada configuração, sob diferentes cargas de trabalho. Em seguida, analisaremos os resultados obtidos para cada uma das configurações, considerando os dois cenários de implantação, primeiramente em separado e depois conjuntamente.

4.1. Configurações de 1 VM

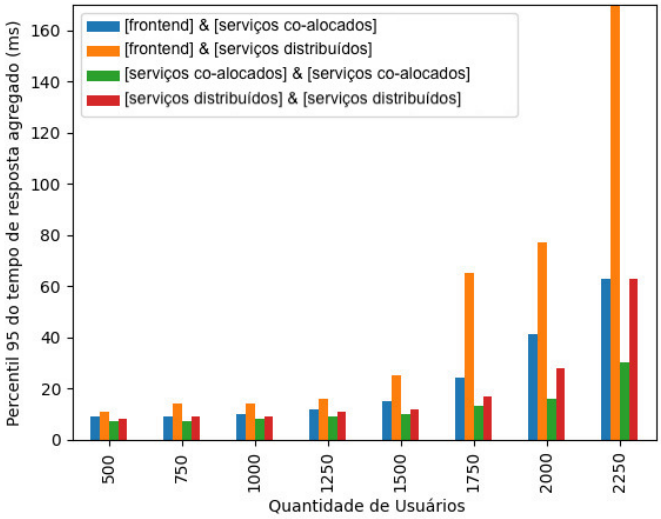
A Figura 3(a) ilustra o desempenho das quatro configurações de implantação em uma única VM. A configuração monolítica mostrou um aumento progressivo no tempo de resposta, de 6 ms para 500 usuários a 17 ms para 2250 usuários, indicando limitações de escalabilidade sob cargas maiores devido à sua natureza centralizada. A configuração [frontend & serviços co-alocados], com tempos de resposta aumentando de 6 ms para 21 ms, teve um desempenho inicial semelhante, mas sofreu um aumento ligeiramente mais acentuado sob a carga mais elevada, provavelmente causado pelo maior número de invocações remotas decorrentes do desacoplamento do serviço *frontend* dos demais serviços. A configuração [serviços co-alocados & serviços co-alocados] mostrou uma tendência semelhante às anteriores, com um tempo inicial de 7 ms e aumento para 21 ms, indicando que a divisão dos serviços em dois grupos acoplados oferece algum benefício em termos de distribuição de carga, mas ainda enfrenta desafios semelhantes de escalabilidade devido às invocações remotas entre os dois grupos de serviços. Por fim, a configuração [serviços distribuídos], com tempos variando entre 8 ms e 68 ms, mostrou um desempenho semelhante às demais configurações inicialmente, mas experimentou um aumento significativo na carga mais alta, o maior entre as quatro configurações do cenário com uma única VM. A natureza totalmente desacoplada dos serviços, embora ofereça maior flexibilidade e potencial de escalabilidade inicial, acaba por deterior o desempenho desta configuração devido à sobrecarga de comunicação entre os serviços.

4.2. Configurações de 2 VMs

As configurações em duas VMs (Figura 3(b)) mostraram uma maior variabilidade nos resultados, refletindo a maior complexidade da distribuição dos serviços neste cenário



(a) Desempenho das configurações de uma VM



(b) Desempenho das configurações de duas VMs

Figura 3. Desempenho das oito configurações de implantação da aplicação alvo.

de implantação. A configuração [frontend] & [serviços co-alocados] exibiu um aumento no tempo de resposta de 9 ms para 63 ms, indicando que o isolamento do *frontend* em uma VM separada tornou-se um gargalo em altas cargas. A configuração [frontend] & [serviços distribuídos] iniciou com um tempo de resposta de 11 ms, aumentando drasticamente para 170 ms sob a carga máxima, de longe o pior desempenho entre todas as configurações avaliadas. Este resultado sugere que a complexidade da comunicação entre serviços totalmente desacoplados em VMs diferentes é um fator crítico em altas cargas. A configuração [serviços co-alocados] & [serviços co-alocados] ofereceu a melhor escalabilidade entre as configurações de duas VMs, com um aumento de 7 ms para 30 ms, similar à escalabilidade das melhores configurações de uma VM. A divisão dos serviços em dois grupos acoplados em VMs separadas parece oferecer um bom equilíbrio entre a granularidade dos serviços e a sobrecarga de comunicação entre eles, resultando em um aumento gradual e controlado do tempo de resposta. Por fim, a configuração [serviços

distribuídos] & [serviços distribuídos] teve um desempenho similar ao da configuração [frontend] & [serviços co-alocados], com tempos de resposta aumentando de 8 ms para 63 ms, revelando que a flexibilidade inicial oferecida pelo desacoplamento total entre os serviços nas duas VMs enfrenta desafios significativos de comunicação entre VMs sob altas cargas, embora com um impacto menor que o na configuração desacoplada com o *frontend* isolado na primeira VM.

4.3. Análise Conjunta dos Dois Cenários de Implantação

Comparando o desempenho de todas as configurações, é possível extrair algumas tendências gerais sobre o impacto da granularidade dos serviços e da comunicação entre máquinas. Nas configurações de uma VM, observa-se um aumento progressivo nos tempos de resposta com o aumento da carga, especialmente evidente nas configurações totalmente distribuídas, onde a sobrecarga de comunicação entre processos se torna um fator limitante. Em contraste, as configurações de duas VMs demonstram que a distribuição física dos serviços entre diferentes máquinas pode introduzir um custo adicional de comunicação ente os serviços. Em particular, configurações com serviços totalmente distribuídos em duas VMs sofrem um aumento mais acentuado nos tempos de resposta sob altas cargas, indicando que a comunicação entre processos localizados em uma mesma VM pode ser mais crítica do que a comunicação entre processos localizados em diferentes VMs. Por outro lado, as configurações com serviços co-alocados em duas VMs apresentaram um desempenho semelhante ao das configurações com serviços co-alocados em uma única VM, indicando que a comunicação entre VMs pode ser menos crítica quando os serviços são acoplados.

Estes resultados sugerem que, enquanto a granularidade dos serviços oferece vantagens de escalabilidade em cargas mais baixas, a complexidade e a sobrecarga associadas às invocações remotas, especialmente em configurações distribuídas entre várias VMs, se tornam fatores dominantes sob alta carga. Assim, a escolha entre diferentes configurações de implantação de microsserviços deve considerar um equilíbrio entre a flexibilidade oferecida pela distribuição física dos serviços e as implicações de desempenho associadas às estratégias de comunicação entre processos de uma mesma Vm ou diferentes VMs.

5. Trabalhos Relacionados

O problema de como determinar e avaliar o impacto da granularidade de microsserviços tem sido amplamente investigado na literatura [Shadija et al. 2017, Newman 2019, Homay et al. 2019, Hassan et al. 2020, Vera-Rivera et al. 2021, Costa and Ribeiro 2021]. Segundo a revisão sistemática da literatura realizada por [Vera-Rivera et al. 2021], a maioria dos trabalhos nesse tema se concentra na migração de sistemas monolíticos para microsserviços, com pouca atenção dada ao desenvolvimento do zero de sistemas baseados em microsserviços ou à avaliação da granularidade em sistemas existentes, que é o foco deste trabalho. Nesse aspecto, os trabalhos mais relacionados ao nosso são os de [Shadija et al. 2017] e [Costa and Ribeiro 2021], discutidos a seguir.

O trabalho de [Shadija et al. 2017] investiga o efeito da co-localização de serviços na latência de uma aplicação de admissão acadêmica, simulando duas abordagens de implantação: uma com todos os serviços em um único contêiner e outra com os serviços distribuídos em dois contêineres. Os resultados mostraram um aumento negligenciável

na latência para a implantação em dois contêineres em comparação com um único container. Já o trabalho de [Costa and Ribeiro 2021] compara o desempenho de uma aplicação de comércio eletrônico desenvolvida seguindo uma abordagem monolítica tradicional e duas versões distribuídas equivalentes baseadas em diferentes padrões comunicação de microsserviços: comunicação direta e comunicação dirigida a eventos. Os resultados mostraram que a abordagem monolítica apresentou a menor latência em todos os níveis de carga avaliados, enquanto as abordagens distribuídas de comunicação direta e dirigida a eventos apresentaram os piores desempenhos, nessa ordem.

Nosso trabalho difere dos dois trabalhos descritos acima em vários aspectos. Primeiro, enquanto o trabalho de [Shadija et al. 2017] compara o desempenho de duas configurações de implantação da aplicação alvo, em um e dois contêineres, o nosso trabalho compara o desempenho de oito configurações de implantação da aplicação alvo, em uma e duas máquinas virtuais. Segundo, enquanto o trabalho de [Costa and Ribeiro 2021] compara o desempenho de uma aplicação monolítica com o de duas versões distribuídas equivalentes utilizando diferentes padrões de arquitetura, nosso trabalho compara o desempenho de uma versão monolítica da aplicação alvo com o de sete versões distribuídas equivalentes, utilizando a mesma topologia de comunicação entre os microsserviços. Por fim, ambos os trabalhos limitaram-se a avaliar níveis de granularidades de serviços que foram pré-determinados em tempo de projeto das aplicações, enquanto o nosso trabalho se beneficia do arcabouço Service Weaver para avaliar múltiplos níveis de granularidade, definidos em tempo de implantação a partir do acoplamento de diferentes subconjuntos de componentes da aplicação.

6. Conclusão e Trabalhos Futuros

Este trabalho investigou o impacto da granularidade de implantação de microsserviços no desempenho da aplicação distribuída Online Boutique, utilizando o arcabouço Service Weaver. O desempenho da aplicação foi analisado sob múltiplas configurações de implantação e cargas de trabalho, em cenários com uma e duas máquinas virtuais. Os resultados indicaram que a granularidade de implantação e a topologia de comunicação entre os microsserviços têm um impacto significativo no desempenho e escalabilidade da aplicação, destacando o desafio e a importância de se obter um equilíbrio adequado entre esses fatores.

Nosso trabalho é parte de um esforço em andamento para explorar configurações ótimas de granularidade em microsserviços e, por isso, ainda apresenta limitações, como a avaliação com base em uma única aplicação e padrão de comunicação, em apenas duas máquinas virtuais. Futuras extensões podem incluir o teste de outras aplicações e padrões arquiteturais, além de explorar configurações de implantação em ambientes de nuvem com múltiplas VMs. Outro linha de pesquisa a ser explorada é realizar uma análise mais detalhada da correlação de utilização dos microsserviços para a carga de trabalho utilizada nos testes. Isso poderia prover mais informações sobre alternativas potencialmente mais adequadas de organização tanto dos microsserviços como da sua implantação na infraestrutura utilizada nos experimentos. Por fim, a avaliação do impacto da granularidade de microsserviços em outros atributos de qualidade de software, como disponibilidade e resiliência, considerando o uso de padrões como *Retry* e *Circuit Breaker* para lidar com potenciais falhas de comunicação entre os serviços distribuídos, também é um aspecto relevante a ser investigado em trabalhos futuros.

Agradecimentos

Os assistentes de inteligência artificial gerativa ChatGPT-4 e GitHub Copilot foram utilizados na estruturação e revisão de várias seções deste artigo.

Referências

Campbell, M. (2023). Google Service Weaver Enables Coding as a Monolith and Deploying as Microservices. https://www.infoq.com/news/2023/03/google-weaver-framework/. [Último acesso em 19 de abril de 2024].

Costa, L. and Ribeiro, A. N. (2021). Performance Evaluation of Microservices Featuring Different Implementation Patterns. In *International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 165–176. Springer.

Cui, J., Chen, P., and Yu, G. (2020). A Learning-based Dynamic Load Balancing Approach for Microservice Systems in Multi-cloud Environment. In *IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 334–341. IEEE.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow. In *Present and Ulterior Software Engineering*. Springer.

Ghemawat, S., Grandl, R., Petrovic, S., Whittaker, M., Patel, P., Posva, I., and Vahdat, A. (2023). Towards Modern Development of Cloud Applications. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems*, pages 110–117.

Google (2024). Service Weaver: A Programming Framework for Writing and Deploying Cloud Applications. https://serviceweaver.dev. [Último acesso em 19 de abril de 2024].

Google Cloud (2023). Online Boutique. https://github.com/GoogleCloudPlatform/microservices-demo. [Último acesso em 19 de abril de 2024].

Hassan, S., Bahsoon, R., and Kazman, R. (2020). Microservice transition and its granularity problem: A systematic mapping study. *Software: Practice and Experience*, 50(9):1651–1681.

Homay, A., Zoitl, A., de Sousa, M., Wollschlaeger, M., and Chrysoulas, C. (2019). Granularity Cost Analysis for Function Block as a Service. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 1199–1204. IEEE.

Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., and Tilkov, S. (2018). Microservices: The Journey So Far and Challenges Ahead. *IEEE Software*, 35(3):24–35.

Locust.io (2024). Locust: An open source load testing tool. https://locust.io/. [Último acesso em 19 de abril de 2024].

Molyneaux, I. (2014). *The Art of Application Performance Testing: From Strategy to Tools*. O'Reilly Media.

Newman, S. (2019). *Monolith to Microservices: Evolutionary Patterns to Transform your Monolith*. O'Reilly Media.

Oracle (2024). VirtualBox. https://www.virtualbox.org/. [Último acesso em 19 de abril de 2024].

Park, J., Choi, B., Lee, C., and Han, D. (2021). GRAF: A graph neural network based proactive resource allocation framework for SLO-oriented microservices. In *Proc. of the 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 154–167.

Preston-Werner, T. (2024). TOML: A config file format for humans. https://toml.io/. [Último acesso em 19 de abril de 2024].

Saleh Sedghpour, M. R., Klein, C., and Tordsson, J. (2022). An Empirical Study of Service Mesh Traffic Management Policies for Microservices. In *ACM/SPEC Int. Conf. Performance Engineering (ICPE)*, pages 17–27. AMC.

Shadija, D., Rezai, M., and Hill, R. (2017). Microservices: Granularity vs. Performance. In *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing (UCC)*, pages 215–220.

Soldani, J., Muntoni, G., Neri, D., and Brogi, A. (2021). The μTOSCA toolchain: Mining, analyzing, and refactoring microservice-based architectures. *Software: Practice and Experience*, 51(7):1591–1621.

The Kubernetes Authors (2024). Minikube. https://minikube.sigs.k8s.io/docs/. [Último acesso em 19 de abril de 2024].

Vera-Rivera, F. H., Gaona, C., and Astudillo, H. (2021). Defining and measuring microservice granularity——a literature overview. *PeerJ Computer Science*, 7:e695.