Adoção e Uso de Virtual Threads em Projetos Open-Source Java: Um Estudo Empírico

João Paulo de Sales Pimenta[1]

[1]Instituto de Ciências Exatas e Informática (ICEI)
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Engenharia de Software – Campus Lourdes (Praça da Liberdade)
Rua Alvarenga Peixoto, 159 – Bairro Lourdes – Belo Horizonte – MG – Brasil
CEP: 30180-120

joao.pimenta.1433569@sga.pucminas.br

Abstract. *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

Resumo. *Este meta-artigo descreve o estilo a ser usado na confecção de artigos e resumos de artigos para publicação nos anais das conferências organizadas pela SBC. É solicitada a escrita de resumo e abstract apenas para os artigos escritos em português. Artigos em inglês deverão apresentar apenas abstract. Nos dois casos, o autor deve tomar cuidado para que o resumo (e o abstract) não ultrapassem 10 linhas cada, sendo que ambos devem estar na primeira página do artigo.*

Bacharelado em Engenharia de *Software* - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientador de conteúdo (TCC I): Danilo Maia - dqmf88@yahoo.com.br
Orientador de conteúdo (TCC I): Leonardo Vilela - leonardocardoso@pucminas.br
Orientador de conteúdo (TCC I): Raphael Ramos - rrdcostasi@gmail.com
Orientador acadêmico (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientador do TCC II: (A ser definido no próximo semestre)

Belo Horizonte, 11 de outubro de 2025.

1. Introdução

O desenvolvimento de software moderno é crescentemente marcado por aplicações distribuídas, principalmente por arquiteturas de microsserviços hospedadas em ambientes de computação em nuvem, nas quais a gestão eficiente de CPU e memória é crítica para desempenho e custo operacional [Mohamed et al. 2021]. Historicamente, Java aborda concorrência com *platform threads* (threads de plataforma), que encapsulam threads do sistema operacional e, embora familiares, tendem a ser "pesadas" quando mantidas em grande número, especialmente em cargas intensivas de entrada/saída (E/S) [Navarro et al. 2023, Lašić et al. 2024]. Modelos assíncronos e reativos difundiram-se

nos sistemas para resolver tais limitações por meio de E/S não bloqueante, ao custo de maior complexidade de desenvolvimento e manutenção [Navarro et al. 2023]. Nesse contexto, o JDK introduziu *virtual threads* (threads virtuais, VT) em *preview* e as estabilizou no JDK 21, conciliando programação síncrona com alta concorrência e baixo custo por tarefa [Pressler et al. 2023].

Apesar desse avanço, **não há um panorama empírico consolidado sobre a adoção contemporânea de *virtual threads* em projetos Java: onde aparecem, com que prevalência em relação a *threads* de plataforma e quais padrões concretos de uso predominam em diferentes *frameworks* e domínios**. Embora as VTs estejam estáveis no JDK 21 e sua viabilidade técnica tenha sido demonstrada, ainda faltam evidências sistemáticas sobre a influência de fatores como estilo de comunicação entre processos/serviços (IPC, do inglês *Inter-Process Communication*), versões do JDK e bibliotecas de E/S na decisão de adoção, bem como sobre a interação com configurações padrão de *frameworks* e estilos arquiteturais [Mohamed et al. 2021, Navarro et al. 2023].

A relevância do problema decorre das seguintes questões: as *virtual threads* foram concebidas para promover melhoria de desempenho, eficiência e racionalização de recursos em software, otimizando o uso de CPU e memória em aplicações intensivas em *I/O* e aumentando a taxa de transferência (*throughput*) [Navarro et al. 2023]. Trata-se de uma capacidade estabilizada no JDK 21 [Pressler et al. 2023], sobre a qual ainda há escassez de estudos em cenários realistas, especialmente em aplicações orientadas a banco de dados na nuvem [Lašić et al. 2024]. Em paralelo, há sinais de adoção em *frameworks* e servidores web populares, como Quarkus, Spring e Jetty [Rosà et al. 2023]. Nesse contexto, compreender benefícios, custos e desafios de integração torna-se essencial para embasar decisões técnicas, mitigar riscos e orientar a construção de sistemas escaláveis e eficientes. Além disso, a estabilização das VTs no JDK 21 cria um período oportuno para observar tendências reais de adoção e consolidar recomendações que orientem decisões de engenharia com foco em automação, eficiência e uso de recursos [Pressler et al. 2023].

O objetivo geral deste trabalho é investigar a adoção contemporânea de *virtual threads* em projetos Java, categorizando sua prevalência, padrões de uso e fatores associados, em contraste com o emprego de *threads* de plataforma. Especificamente, pretende-se: (i) minerar repositórios de código abertos para identificar indícios de adoção (por exemplo, ocorrências de `Thread.ofVirtual()` e executores de VT, além de opções de *frameworks* que habilitam VT); (ii) classificar padrões de uso observados (como *thread-per-request*, mapeamentos VTs ou PTs e arranjos híbridos com *pools*); (iii) analisar fatores associados à adoção, incluindo *framework*, versão do JDK, estilo de comunicação entre serviços (APIs *REST* e corretores de mensagens) e bibliotecas de acesso a dados e (iv) examinar a evolução temporal pós-JDK 21 por período, domínio e ecossistema.

Como resultados esperados, tem-se: (a) um panorama verificável da prevalência de VTs versus *threads* de plataforma em projetos Java atuais; (b) uma classificação de padrões de uso de VTs com exemplos de código e contexto arquitetural; (c) um conjunto de fatores correlacionados à adoção (ou não adoção) combinando *frameworks*, versões do JDK e bibliotecas de E/S para orientar decisões de engenharia; e (d) um catálogo de barreiras e antipadrões com sugestões de mitigação. Tais artefatos buscam apoiar a automação (simplificação do código concorrente), a eficiência e a racionalização de re-

cursos no desenvolvimento e na operação de software.

Por fim, este trabalho está organizado da seguinte forma: a Seção 2 apresenta o referencial teórico e os trabalhos relacionados sobre VTs em aplicações Java (incluindo conceitos, relatos de adoção e usos característicos); a Seção 3 descreve o desenho do estudo empírico (estratégia de mineração, critérios de inclusão/exclusão, extração de evidências e métricas); a Seção 4 relata os resultados de adoção, padrões e fatores observados; a Seção 5 discute implicações práticas, ameaças à validade e recomendações de engenharia; e a Seção 6 conclui o trabalho e sugere direções futuras.

2. Trabalhos Relacionados

Os trabalhos apresentados nesta seção estabelecem a viabilidade técnica e os benefícios de desempenho das VTs em cenários intensivos em E/S, como o acesso a dados e a renderização de *templates web*. Estudos demonstram que VTs superam consistentemente as *platform threads* (PTs) tradicionais em eficiência de tempo de execução em tarefas de concorrência e análise de grafos, mas são ligeiramente mais lentas em programação puramente paralela (*CPU-bound*).

Lašić et al. (2024) conduziram um estudo com o objetivo de avaliar a eficiência das *virtual threads* em aplicações de servidor orientadas a banco de dados. O método empregado envolveu a comparação da eficiência das VTs com as PTs tradicionais de Java em aplicações de servidor que utilizam o modelo *thread-per-request* e acessam bancos de dados relacionais (*MySQL, PostgreSQL e Oracle*) e não-relacionais (*MongoDB, Neo4j, Cassandra*). Os resultados preliminares demonstraram que as VTs consistentemente apresentaram desempenho superior ao das *threads* tradicionais, especialmente em operações CRUD de alto *throughput*. Em aplicações com bancos de dados relacionais, a latência foi reduzida em média em aproximadamente 100 microssegundos. Os autores propõem a utilização de VTs em aplicações de servidor modernas baseadas em *frameworks* e orientadas a banco de dados em nuvem. O estudo fornece a evidência técnica de que o uso de VTs traz benefícios claros de desempenho em cenários de E/S intensivos, como o acesso a dados, o que é relevante para o objetivo de investigar a adoção empírica das VTs em relação às bibliotecas de acesso a dados.

Outro trabalho realiza uma análise da eficiência das *virtual threads* em cenários de programação paralela (*CPU-bound*). Sirotic et al. (2025) comparou a velocidade de execução de quatro programas paralelos em Java (*Executors, ForkJoin, Streams e Virtual Executors*) para a tarefa de contagem de números primos. O principal resultado encontrado é que as VTs se mostraram ligeiramente mais lentas do que a solução que utilizava *threads* não virtuais para o problema. Por exemplo, a solução com PTs foi marginalmente mais rápida ($1,54$ s *vs.* $1,56$ s) em um processador mais antigo, e ($0,22$ s *vs.* $0,23$ s) em um processador mais novo. Este dado é crucial para o presente estudo, pois o uso de VTs para tarefas puramente *CPU-bound* é considerado um antipadrão, o que, se observado em projetos reais, ajuda a catalogar barreiras e antipadrões.

Investigando o ambiente de *frameworks* reativos, Navarro et al. (2023) analisaram considerações para a integração de *virtual threads* em um *framework* Java reativo (Quarkus), com foco em ambientes com recursos limitados. Os resultados revelaram que a integração inicial de VTs não apresentou desempenho tão bom quanto a abordagem reativa do *framework*. Esta limitação foi atribuída a uma incompatibilidade fundamental

entre as VTs e o *framework* Netty, especialmente devido à alta pressão sobre o *Garbage Collector* (GC) causada pelo uso de *ThreadLocals*. O estudo de Navarro *et al.* detalha as complexidades e barreiras técnicas que podem influenciar a adoção de VTs em arquiteturas.

Complementarmente, um estudo explorou como as VTs facilitam o *Progressive Server-Side Rendering* (PSSR) utilizando *template engines web* tradicionais que dependem de interfaces bloqueantes. Os autores Pereira et al. (2025) concluíram que as VTs permitem que *engines* bloqueantes atinjam escalabilidade comparável àquelas projetadas para E/S não bloqueante, mantendo alto *throughput* e simplificando a implementação de PSSR com estilos de código síncrono familiar. Em implementações Quarkus, a abordagem de VTs alcançou *throughput* elevado (4856 requisições por segundo) e escalou efetivamente até 128 usuários concorrentes. Este trabalho valida que VTs são uma solução eficaz para o padrão de uso *thread-per-request* em aplicações *web I/O-bound*, um dos padrões que serão investigados.

Chirila e Sora (2024) investigaram a avaliação de desempenho em tempo de execução de *single thread* (ST), PT e VT no contexto da detecção de classes-chave. O objetivo era medir o desempenho da paralelização dos algoritmos *HITS* e *PageRank*, cruciais para a análise de grafos em engenharia de *software*. O método consistiu em executar implementações dos algoritmos usando os três modelos de *threading* (ST, PT e VT) em um conjunto de 14 projetos Java de tamanhos variados. Os algoritmos *HITS* e *PageRank* foram aplicados na representação em grafos dos sistemas analisados, ranqueando as classes por sua importância. O estudo revelou que o modelo VT superou consistentemente os modelos ST e PT, resultando em uma diminuição de tempo de execução de 58,41% em comparação com o modelo ST no conjunto total de projetos. Eles também observaram que o modelo ST é melhor do que o PT em 12 dos 14 projetos, e o PT só superou o ST em projetos com mais de 1.700 nós, que consomem 90% do tempo de execução total do conjunto. Este trabalho estabelece um contexto de desempenho geral, demonstrando a maior eficiência das VTs sobre as PTs, fornecendo um forte argumento para a adoção.

Por fim, um trabalho empírico que servirá de base metodológica para a mineração de código proposta é o de Zimmerle et al. (2022), que se concentrou na mineração do uso de APIs de *Reactive Programming* (RP) em projetos *open-source*. O objetivo era entender a prevalência do uso dos operadores em três bibliotecas *Reactive Extensions* (Rx) (RxJava, RxJS e RxSwift). O método empregado envolveu a mineração de repositórios no GitHub que possuíam pelo menos 10 estrelas, utilizado como um filtro de popularidade. Os autores baixaram os projetos e utilizaram expressões regulares (*regex*) para buscar a ocorrência de invocações de operadores. O estudo demonstrou que $95,2\%$ dos operadores Rx estavam em uso, validando a mineração de repositórios como uma abordagem eficaz para quantificar a adoção e o uso de APIs específicas em grande escala. O presente estudo adotará uma abordagem de mineração similar para identificar a adoção de VTs, por exemplo, ocorrências de *Thread.ofVirtual()* e *executors* de VT como instâncias criadas por *Executors.newVirtualThreadPerTaskExecutor()*) em projetos *open-source* Java.

Referências

Chirilă, C. and Şora, I. (2024). Java single vs. platform vs. virtual threads runtime performance assessment in the context of key class detection. In *2024 IEEE 18th Inter-*

national Symposium on Applied Computational Intelligence and Informatics (SACI). IEEE.

Lašić, L., Beronić, D., Mihaljević, B., and Radovan, A. (2024). Assessing the efficiency of Java virtual threads in database-driven server applications. In *47th MIPRO ICT and Electronics Convention (MIPRO)*, pages 2045–2050. IEEE.

Mohamed, H., Khazaei, S., Khazaei, H., and Taherkordi, A. (2021). End-to-end latency prediction of microservices workflow on Kubernetes: A comparative evaluation of machine learning models and resource metrics. In *Proceedings of the 54th Hawaii International Conference on System Sciences (HICSS-54)*, pages 1717–1726. HICSS.

Navarro, A., Ponge, J., Mouël, F. L., and Escoffier, C. (2023). Considerations for integrating virtual threads in a Java framework: a Quarkus example in a resource-constrained environment. In *Proceedings of the 17th ACM International Conference on Distributed and Event-Based Systems (DEBS '23) Companion*, Neuchâtel, Switzerland. ACM.

Pereira, B. and Carvalho, F. M. (2025). Enabling progressive server-side rendering for traditional web template engines with java virtual threads. *Software*, 4(3):20.

Pressler, R., Bateman, A., and Reinhold, M. (2023). Jep 444: Virtual threads. `https://openjdk.org/jeps/444`.

Rosà, A., Basso, M., Bohnhoff, L., and Binder, W. (2023). Automated runtime transition between virtual and platform threads in the Java virtual machine. In *30th Asia-Pacific Software Engineering Conference (APSEC)*, pages 607–611. IEEE.

Sirotić, Z., Sovilj, S., Oršulić, M., and Pripužić, K. (2025). Comparison of Java virtual and non-virtual threads in parallel programming. In *48th MIPRO ICT and Electronics Convention (MIPRO), Software and Systems Engineering Track (SSE)*. IEEE.

Zimmerle, C., Gama, K., Castor, F., and Filho, J. M. M. (2022). Mining the usage of reactive programming apis: A study on github and stack overflow. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 203–214.