

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO
SUDESTE DE MINAS GERAIS - CAMPUS RIO POMBA**

STEPHANYE CRISTINE ANTUNES DE CUNTO

ESCREVER O TÍTULO AQUI

RIO POMBA - MG

2025

STEPHANYE CRISTINE ANTUNES DE CUNTO

ESCREVER O TÍTULO AQUI

Trabalho de Conclusão de curso apresentado ao *Campus* Rio Pomba, do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais, como parte das exigências do curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel em Ciência da Computação.

Orientadora: ME. BIANCA PORTES DE CASTRO

Coorientador: DR. JOSÉ RUI CASTRO DE SOUSA

RIO POMBA - MG

2025

FICHA CATALOGRÁFICA TEMPORÁRIA
STEPHANYE CRISTINE ANTUNES DE CUNTO
ESCREVER O TÍTULO AQUI/ STEPHANYE CRISTINE ANTUNES DE
CUNTO. – RIO POMBA - MG, 2025-
Orientadora: ME. BIANCA PORTES DE CASTRO

Trabalho de Conclusão de Curso – Instituto Federal de Educação,
Ciência e Tecnologia do Sudeste de Minas, Campus Rio Pomba

STEPHANYE CRISTINE ANTUNES DE CUNTO

ESCREVER O TÍTULO AQUI

Trabalho de Conclusão de curso apresentado ao *Campus* Rio Pomba, do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais, como parte das exigências do curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho aprovado em XX de XXXXX de XXXX.

ME. BIANCA PORTES DE CASTRO
Orientadora, IF Sudeste MG - Rio Pomba

DR. JOSÉ RUI CASTRO DE SOUSA
Coorientador, IF Sudeste MG - Rio Pomba

TÍTULO E NOME DO MEMBRO DA BANCA
IF Sudeste MG - Rio Pomba

RIO POMBA - MG

2025

*“Os olhos não são apenas
o espelho da alma,
mas também do corpo.”
(Ignatz von Peczelly, 1989)*

Resumo

O resumo é um texto breve que apresenta, de forma clara e objetiva, os principais elementos da monografia. Ele deve permitir que o leitor compreenda rapidamente sobre o que é o trabalho, qual foi a abordagem adotada e quais foram os resultados e conclusões. Na ABNT (NBR 6028), recomenda-se que o resumo seja escrito em parágrafo único, sem subdivisões e sem citações diretas, geralmente com 150 a 500 palavras. O que incluir no resumo: Tema e objetivo Apresente o assunto principal e o objetivo geral do trabalho; Metodologia Informe de forma resumida o método, técnicas ou procedimentos utilizados; Resultados Destaque os resultados mais relevantes obtidos na pesquisa; Conclusão Apresente a principal contribuição ou conclusão do estudo. Dicas importantes: Escreva no tempo passado, já que o trabalho foi realizado. Evite usar abreviações pouco conhecidas ou siglas sem explicação. Não insira informações que não estejam no corpo do trabalho. Revise para garantir clareza, coerência e objetividade. Lembre-se de incluir palavras-chave logo abaixo do resumo (entre três e cinco termos que representem bem o conteúdo do trabalho, separados com ponto e vírgula). Importante: O resumo deve ser escrito após a conclusão do trabalho, quando todos os resultados já estão definidos. Assim, será fiel ao conteúdo final da monografia.

Palavras-Chave:

Abstract

Tradução do resumo para a língua inglesa.

Key-words:

Lista de abreviaturas e siglas

CPU	Unidade Central de Processamento
I/O	Entrada/Saída
JVM	Java Virtual Machine (Máquina Virtual Java)
SO	Sistema Operacional
SPD	Sistemas Paralelos E Distribuídos
XXX	INSERIR EM ORDEM ALFABÉTICA

Sumário

1	Introdução	9
2	Fundamentação Teórica	11
2.1	Para criar um título	11
2.1.1	mais um título	11
2.1.1.1	O que escrever na Fundamentação Teórica	11
3	Trabalhos Relacionados	13
3.1	Benchmarking the Performance of Java Virtual Threads in High-Throughput Workloads	13
3.2	Uma análise comparativa entre threads e green threads no Java	13
3.3	Comparison of Concurrency Technologies in Java	14
3.4	Avaliação dos mecanismos de concorrência na API do Java 8	14
3.5	Síntese dos Trabalhos Relacionados	15
4	Metodologia	16
4.1	Ambiente de Teste	16
4.2	Implementação do Servidor	18
4.3	Teste de Carga Constante	19
4.4	Teste de carga com aumento gradual (<i>ramp up</i>)	20
4.5	Controles Experimentais	21
4.6	Métricas Coletadas	21
4.7	Tratamento de erros	22
4.8	Análise Estatística	22
4.9	Limitações dos Testes	24
5	Aplicação / Implementação / Experimento	25
6	Conclusão	26
	Referências	27

1 Introdução

Uma thread é a menor unidade de processamento existente dentro de um processo. Cada processo é capaz de conter múltiplas threads, permitindo a execução simultânea de diferentes partes de um programa ou de diferentes tarefas (Silberschatz; Galvin; Gagne, 2018).

A utilização de threads oferece diversos benefícios em problemas que envolvem tarefas de alto processamento (*CPU-bound*) e operações de entrada e saída (*I/O-bound*). Além disso, em sistemas reativos e servidores, threads permitem que aplicações atendam múltiplos usuários simultaneamente, mantendo respostas rápidas e contínuas mesmo sob alta carga, garantindo desempenho eficiente e melhor experiência ao usuário. Assim, o uso de threads é aplicado como estratégia para otimizar recursos computacionais e melhorar a eficiência de sistemas em diferentes contextos.

O relatório *State of the Octoverse 2024* (GitHub, 2024) demonstra que a linguagem de programação Java está entre as cinco linguagens mais utilizadas na plataforma, o que reforça sua importância como uma das principais tecnologias do desenvolvimento de software.

No lançamento da versão 19 do Java, foram introduzidas as threads virtuais, que, diferente das threads tradicionais, são gerenciadas pela *Java Virtual Machine* (JVM). Enquanto o escalonamento das threads tradicionais é realizado pelo sistema operacional (SO), determinando quando cada thread é executada, o escalonamento das threads virtuais é feito pela própria JVM, podendo apresentar comportamentos distintos (Oracle Corporation, 2023).

O uso de threads tradicionais pode gerar *overhead* e limitar a escalabilidade. Dessa forma, as threads virtuais surgem como alternativa para contornar essas limitações. Este trabalho investiga como a utilização de threads virtuais impacta o desempenho e a escalabilidade de aplicações concorrentes.

O objetivo deste trabalho é analisar as diferenças de desempenho entre threads tradicionais e threads virtuais, avaliando como cada abordagem impacta a execução de aplicações concorrentes.

Para alcançar o objetivo geral, foram definidos os seguintes objetivos específicos:

- Compreender os fundamentos teóricos da programação concorrente e das diferentes abordagens de threads;
- Implementar protótipos de aplicações concorrentes utilizando ambas as abordagens;
- Medir e comparar métricas de desempenho;

- Analisar os resultados obtidos, identificando vantagens e limitações.

A escolha deste tema parte do interesse gerado durante a disciplina de Sistemas Paralelos e Distribuídos (SPD), na qual foram estudados os conceitos fundamentais de concorrência, paralelismo e gerenciamento de threads. Além disso, o apreço pessoal pela linguagem Java contribuiu para a definição do tema, considerando que a plataforma tem investido em melhorias relacionadas à programação concorrente. Dessa forma, esse trabalho se justifica pelo interesse acadêmico adquirido ao longo da disciplina e pela pertinência de investigar uma inovação recente da linguagem Java, contribuindo para a compreensão de como esse novo modelo de threads pode aprimorar o desempenho de aplicações concorrentes.

Este trabalho está organizado da seguinte forma:

- **Seção 1 - Introdução:** Apresenta o tema, os objetivos e a justificativa.
- **Seção 2 - Fundamentação teórica:** Expõe os conceitos de concorrência e paralelismo e descreve o funcionamento de processos, threads tradicionais e threads virtuais.
- **Seção 3 - Trabalhos Relacionados:** Revisa estudos anteriores sobre o desempenho de threads tradicionais e threads virtuais.
- **Seção 4 - Metodologia:** Detalha como os testes foram conduzidos e como os dados foram coletados.
- **Seção 5 - Resultados e Discussão:** Apresenta e analisa os dados obtidos durante os testes.
- **Seção 6 - Conclusão:** Expõe as considerações finais do estudo.

Por fim, são listadas as referências bibliográficas utilizadas no trabalho, seguidas pelos apêndices e anexos que complementam o estudo.

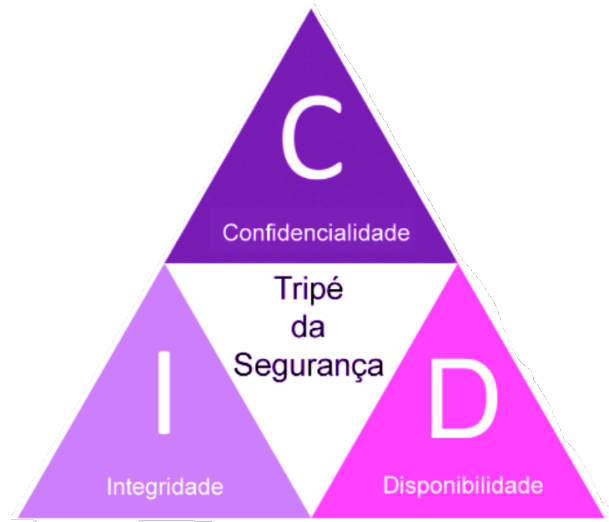
2 Fundamentação Teórica

Exemplo de citação no final do texto (PECB, 2022).

Exemplo de citação dentro do texto PECB (2022).

Exemplo de uma Figura. Use ref para chamá-la no texto. Figura 1.

Figura 1 – Pilares da Segurança da Informação.



Fonte: Bughunt (2023).

2.1 Para criar um título

Sempre inserir um texto entre os Títulos

2.1.1 mais um título

Um exemplo de Quadro (Quadro 1).

Categoria	Resultado
Frequência	3.661 incidentes, sendo (82,8%) com vazamento confirmado de dados.
Atores de ameaça	(100%) externos (<i>breaches</i>).
Motivações	(95%) financeiras, (5%) espionagem (<i>breaches</i>).
Dados comprometidos	Credenciais (50%), pessoais (41%), internos (20%), outros (14%).

Fonte: Verizon (2024), adaptado.

Um exemplo de Tabela.

2.1.1.1 O que escrever na Fundamentação Teórica

A Fundamentação Teórica é a base conceitual do seu trabalho. Nela, você apresenta, discute e analisa as teorias, conceitos, modelos e estudos já existentes que sustentam a sua

Tabela 1 – Notificações formais de incidentes e vulnerabilidades em órgãos públicos entre 2021 e 2025.

Ano	Vulnerabilidades	Incidentes	Total de notificações
2025	1994	4859	6853
2024	5115	9803	14918
2023	10225	4905	15130
2022	5128	3402	8530
2021	4964	4903	9867
Total	27426	27872	55298

Fonte: CTIR Gov (2025).

pesquisa.

Incluir conceitos e definições Explique os principais termos, conceitos e elementos que serão usados no trabalho, sempre com referência a autores da área.

Modelos, teorias e abordagens Traga as principais correntes teóricas que embasam seu estudo.

Dicas importantes:

- Sempre cite as fontes de onde retirou as informações (seguindo as normas da ABNT).
- Organize o texto de forma lógica, por temas ou subtemas, evitando apenas listar autores.
- Não copie trechos longos; prefira escrever com suas palavras e citar corretamente.
- Evite incluir opiniões pessoais mantenha o foco no que já foi publicado por outros autores.

Relacionamento com o seu trabalho Mostre como essas teorias e estudos se aplicam ou se relacionam com a sua pesquisa.

3 Trabalhos Relacionados

Esta seção apresenta trabalhos que servem de base para a pesquisa sobre threads tradicionais e threads virtuais em Java. São discutidos estudos anteriores que investigaram o desempenho dessas threads. Com a revisão realizada, é possível compreender as abordagens utilizadas, os resultados obtidos e as lacunas existentes, fornecendo um referencial teórico sólido para os testes conduzidos neste trabalho.

3.1 Benchmarking the Performance of Java Virtual Threads in High-Throughput Workloads

Pandita (2024) apresentou uma comparação entre threads virtuais e threads tradicionais, considerando escalabilidade, utilização de recursos e latência em cargas de trabalho de alto rendimento. Foram realizados dois benchmarks: CPU-bound, envolvendo cálculos de números primos, e I/O-bound, simulando operações de bloqueio entre duas aplicações Java em ambiente AWS EC2.

Os resultados mostraram que, em tarefas com baixo uso de CPU, ambos os modelos apresentam desempenho semelhante. Em cenários intensivos de CPU, a limitação dos recursos computacionais restringe o aumento da simultaneidade. Por outro lado, em operações I/O-bound, threads virtuais se destacaram, oferecendo maior simultaneidade, redução do consumo de memória e melhor aproveitamento da CPU. Esses dados reforçam a eficiência das threads virtuais em operações de I/O e fornecem um referencial relevante para estudos comparativos, como o presente trabalho.

3.2 Uma análise comparativa entre threads e green threads no Java

Souto (2024) realizou uma pesquisa comparando o desempenho de threads tradicionais e threads virtuais. O estudo analisou operações fundamentais de gerenciamento de threads, o tempo necessário para instanciá-las, iniciá-las, finalizá-las e realizar a troca de contexto em ambas. Para isso, foram utilizados testes com diferentes quantidades de threads e a biblioteca JMH do JDK para coleta precisa de tempos de execução.

O estudo demonstrou que, nos cenários testados, as threads virtuais podem ser mais de 100 vezes mais rápidas que as threads tradicionais, evidenciando sua eficiência em operações de criação, inicialização, finalização e mudança de contexto. Apesar do alto desempenho observado, o estudo se concentrou em cenários específicos, abrindo espaço para pesquisas adicionais que explorem diferentes cargas de trabalho e padrões de uso de threads, como os apresentados neste trabalho.

3.3 Comparison of Concurrency Technologies in Java

Em seu trabalho, Gustafsson e Persson (2024) realizou quatro benchmarks com o objetivo de comparar o desempenho entre threads tradicionais, threads virtuais e Reactive Framework em testes de I/O-bound (simulado), CPU-bound e testes mistos.

No teste I/O-bound, utilizou-se o mecanismo `Thread.sleep(100 ms)` para simular tempos de espera. No teste CPU-bound, realizou-se a multiplicação de matrizes 200x200. Os dois testes mistos combinaram cálculos de matrizes e pausas: um focado em I/O-bound multiplicando matrizes 50x50 com pausa de 100 ms, e outro focado em CPU-bound multiplicando matrizes 200x200 com pausa de 50 ms.

Os resultados indicaram que, no teste CPU-bound, threads tradicionais apresentaram melhor desempenho em termos de throughput e latência. No teste I/O-bound, o Reactive Framework se destacou, mostrando maior escalabilidade e menor uso de CPU e memória. Nos testes mistos, threads virtuais obtiveram alta taxa de requisições e baixa latência no cenário CPU-bound, mantendo desempenho elevado no cenário I/O-bound, enquanto threads tradicionais apresentaram maior latência e menor throughput. O que torna nótório que a eficácia das threads virtuais em cenários mistos, evidenciando seu potencial em aplicações que demandam simultaneidade e eficiência de recursos.

3.4 Avaliação dos mecanismos de concorrência na API do Java 8

Uma análise comparativa do desempenho de single thread, threads, `ExecutorService` e `Fork/Join` em algoritmos de ordenação foi realizada por Aguas (2015), utilizando os algoritmos Quicksort, Merge-sort e Pigeonhole Sort. Os testes foram conduzidos em três máquinas com processadores diferentes (Dual-Core, i5 e i7), e cada algoritmo foi executado 60 vezes para cada abordagem de concorrência.

O Quicksort executado com `ForkJoin-Pool` em uma máquina com 8 processadores lógicos apresentou o melhor desempenho em termos de tempo. De forma geral, Quicksort combinado com `ForkJoin-Pool` registrou os menores tempos na maioria dos cenários testados. O estudo demonstrou que a máquina com maior número de processadores lógicos obteve os melhores resultados, comprovando que a execução paralela se beneficia diretamente da disponibilidade de mais núcleos para distribuir as tarefas simultaneamente.

Esse trabalho evidencia a importância da escolha adequada de técnicas de concorrência e da arquitetura de hardware para otimização do desempenho em algoritmos paralelos.

3.5 Síntese dos Trabalhos Relacionados

Observa-se na literatura um crescente enfoque em threads virtuais e tradicionais em Java.

Em seu texto Souto (2024) concentrou-se na análise das operações de criação, inicialização, finalização e troca de contexto, onde se nota que threads virtuais podem ser mais de 100 vezes mais rápidas que threads tradicionais em cenários específicos. Por sua vez Pandita (2024) aprofunda essa análise ao avaliar o desempenho em cargas CPU-bound e I/O-bound, onde nota-se que, em tarefas com baixo uso de CPU, o desempenho é semelhante, porém threads virtuais se destacam em operações I/O, oferecendo maior simultaneidade, redução do consumo de memória e melhor aproveitamento da CPU.

Gustafsson e Persson (2024) expandiu a análise incluindo o Reactive Framework e cenários mistos, mostrando que threads virtuais apresentam alta taxa de requisições e baixa latência em cenários combinados de CPU e I/O, enquanto threads tradicionais apresentam limitações de throughput e latência. Enquanto Aguas (2015) avaliou abordagens de concorrência na execução de algoritmos de ordenação, deixando claro que a execução paralela se beneficia diretamente do aumento do número de núcleos, principalmente quando combinada com ForkJoin-Pool, o que demonstra a importância do alinhamento entre técnicas de concorrência e arquitetura de hardware.

A comparação desses estudos permite identificar padrões: threads virtuais oferecem vantagens em simultaneidade e eficiência de recursos, especialmente em operações de I/O ou cenários mistos, enquanto threads tradicionais ainda podem apresentar desempenho competitivo em tarefas CPU-bound simples.

O presente trabalho se apoia nesses estudos como referencial para analisar o desempenho de threads tradicionais e virtuais em Java, destacando diferenças de comportamento em operações básicas de criação, execução e finalização de threads. Dessa forma, busca-se fornecer uma compreensão prática sobre a eficiência e o uso de recursos dessas duas abordagens de concorrência em cenários controlados.

4 Metodologia

Essa seção descreve a metodologia adotada para comparar o desempenho entre *threads* tradicionais e *threads* virtuais em Java. Os experimentos foram baseados no trabalho de Gustafsson e Persson (2024), como adaptações para o contexto e ambiente deste estudo.

A abordagem envolveu a implementação de um servidor HTTP simples utilizando arquitetura cliente-servidor, onde o servidor foi configurado para utilizar *threads* tradicionais e *threads* virtuais. Dois tipos de testes foram conduzidos: testes de carga constante para avaliar estabilidade ao longo do tempo, e testes de aumento gradual para identificar limites de escalabilidade. As seções seguintes detalham a metodologia adotada.

Os experimentos foram projetados para comparar o comportamento de *threads* tradicionais e *threads* sob condições controladas, focando em desempenho (latência e throughput), consumo de recursos (CPU e memória) e escalabilidade (capacidade de processamento simultâneo).

Para garantir condições equivalentes entre os dois mecanismos de *threads* os experimentos foram conduzidos nas seguintes condições: mesma infraestrutura de hardware, mesmas configurações de sistema operacional, mesma implementação de servidor e mesma metodologia de geração de carga. A ordem de execução foi alternada entre *threads* tradicionais e virtuais para minimizar vieses relacionados ao aquecimento da JVM ou estado do sistema.

Todos os experimentos seguiram um protocolo automatizado que incluía fases de aquecimento (*warmup*), coleta manual de lixo (*garbage collection*) e pausas entre testes. Esse protocolo, detalhado nas seções subsequentes, teve como objetivo isolar o impacto do mecanismo de *threads* de outros fatores que poderiam influenciar os resultados.

4.1 Ambiente de Teste

Nesta seção, são descritas as características do ambiente utilizado para a realização dos experimentos, incluindo a configuração de hardware, software e rede.

Os testes utilizaram uma arquitetura cliente-servidor, como duas máquinas conectadas por uma rede Wi-fi local. A máquina servidor hospedava a aplicação Java, enquanto a máquina cliente era responsável por gerar carga e coletar métricas de desempenho.

Ambas as máquinas foram dedicadas exclusivamente à execução dos experimentos, sem aplicações ou serviços concorrentes em execução, reduzindo interferências no uso de CPU, memória e rede.

A máquina servidora possuía as seguintes características:

- Processador: Apple M2 (8 núcleos)
- Memória RAM: 8 GB
- Sistema Operacional: macOS Tahoe (versão 26)
- Java Development Kit (JDK): versão 21 LTS
- Spring Boot: versão 3.3.2 (obs.: verificar versão)
- Conexão de rede: Wi-fi 50 Mbps

A máquina cliente possuía as seguintes especificações:

- Processador: Intel Core i5-2410M (2 núcleos, 4 threads)
- Memória RAM: 8 GB
- Sistema Operacional: Ubuntu 22.04 LTS
- Ferramenta de carga: Vegeta (versão 12.13.0)
- Conexão de rede: Wi-fi 50 Mbps

Todos os experimentos foram realizados com a comunicação entre servidor e cliente por rede Wi-fi o que pode introduzir variações adicionais de latência. Entretanto, como ambos os mecanismos de threads foram avaliados sob as mesmas condições, esse impacto é considerado equivalente para fins comparativos.

Para evitar que o sistema operacional se tornasse um gargalo durante os experimentos, foram aplicadas configurações específicas em ambas as máquinas. Essas configurações visaram garantir que limitações observadas estivessem associadas ao mecanismo de gerenciamento de threads, e não a restrições impostas pelo sistema operacional ou pela pilha de rede.

- Número máximo global de arquivos abertos no sistema (1048576), cada conexão TCP consome um descritor de arquivo o que poderia limitar o número de conexões simultâneas.
- Número máximo de arquivos abertos por processo (1048576), garantindo que o servidor possa abrir muitos arquivos e conexões de rede simultaneamente.
- Tamanho máximo da fila de conexões pendentes (4096), permitindo que o servidor aceite um grande número de conexões simultâneas.

- Tamanho máximo do buffer de socket (8 MB), otimizando a transferência de dados pela rede.
- Número máximo de processos por usuário (2000), garantindo que o servidor possa criar muitas threads conforme necessário.
- Tamanho do buffer de envio e recebimento TCP (2 MB), otimizando a transferência de dados pela rede.
- Tempo que conexões TCP permanecem em estado TIME_WAIT (250 ms), liberando recursos mais rapidamente.
- Desativação do ACK atrasado do TCP, reduzindo a latência em conexões de rede.
- Faixa de portas TCP (1024 a 65.535), evita esgotamento de portas efêmeras.

As configurações da JVM foram mantidas em seus valores padrão para refletir cenários reais de uso.

4.2 Implementação do Servidor

Foi desenvolvido um servidor HTTP simples utilizando o framework Spring Boot. A escolha do Spring Boot se deve à facilidade de uso para a criação rápida de aplicações web em Java e sua popularidade, sendo listado entre os frameworks web mais populares em pesquisas, como a Stack Overflow (2025).

O servidor foi implementado de forma a suportar threads tradicionais e threads virtuais, permitindo a comparação direta entre os dois mecanismos.

Para evitar que o servidor se tornasse um gargalo durante os experimentos, foram aplicadas configurações no Spring Boot que permitem alto nível de concorrência:

- `server.tomcat.max-threads=50000`: Define o número máximo de threads que o servidor pode criar para processar requisições.
- `server.tomcat.accept-count=20000`: Define o tamanho da fila de conexões pendentes.
- `server.tomcat.connection-timeout=5000`: Define o tempo máximo em milissegundos que o servidor espera por uma conexão.
- `spring.threads.virtual.enabled=true`: Habilita o uso de threads virtuais no Spring Boot.

O servidor expõe três endpoints REST:

- `/threads/traditional`: Processa requisições utilizando threads tradicionais do sistema operacional. Cada requisição inicia uma thread que aguarda 100 ms antes de responder com status HTTP 200.
- `/threads/virtual`: Processa requisições utilizando threads virtuais do Project Loom. A lógica de processamento é idêntica ao endpoint tradicional, diferindo apenas no mecanismo de threading subjacente.
- `/threads/gc`: Endpoint auxiliar para executar manualmente o processo de garbage collection.

A espera de 100 ms foi escolhida para simular operações I/O-bound típicas de aplicações web reais, como consultas a bancos de dados ou chamadas a serviços externos. Esse tempo de bloqueio permite avaliar como cada mecanismo de threads lida com alta concorrência em cenários onde threads frequentemente aguardam operações de entrada/saída.

4.3 Teste de Carga Constante

Esse teste possui o objetivo de avaliar a estabilidade e o desempenho dos mecanismos de *threads* sob uma carga constante ao longo do tempo. A carga foi mantida em um nível fixo, permitindo observar como cada mecanismo lida com a concorrência e o consumo de recursos em um cenário estável.

O teste foi realizado da seguinte maneira:

1. Três séries de aquecimento (*warmup*) com 300 requisições por segundo durante 60 segundos.
2. Aquecimento (*warmup*) com 1000 requisições por segundo durante 120 segundos.
3. Pausa de 60 segundos.
4. Coleta de lixo (*garbage collection*) manual na máquina servidor.
5. Pausa de 20 segundos.
6. Teste principal com 1000 requisições por segundo durante 600 segundos.
7. Pausa de 60 segundos.
8. Repetição dos passos 1 a 7 para o outro mecanismo de threads, em uma nova instância da JVM.

A taxa de 1.000 requisições/segundo foi definida após testes preliminares que identificaram esse valor foi definido após testes preliminares que indicaram que a máquina servidor começava a apresentar erros de conexão acima dessa taxa. A duração de 10 minutos permite observar tendências de longo prazo, como degradação gradual de desempenho ou acúmulo de recursos.

O teste foi repetido 45 vezes para cada mecanismo de threads, totalizando 90 execuções. Cada execução completa (ambos os mecanismos) durou aproximadamente 35 minutos, resultando em aproximadamente 27 horas de experimentação total.

4.4 Teste de carga com aumento gradual (*ramp up*)

O objetivo desse teste foi identificar o ponto em que o servidor começa a apresentar erros de conexão, indicando que atingiu seu limite máximo de requisições processáveis.

O teste foi conduzido da seguinte maneira:

1. Warmup:

- a) Aquecimento com 300 requisições por segundo durante 60 segundos.
- b) Pausa de 60 segundos.
- c) Coleta de lixo (*garbage collection*) manual na máquina servidor.
- d) Pausa de 20 segundos.
- e) Repetição dos passos acima mais duas vezes, com a taxa ajustada para 1000 requisições por segundo (limite seguro do servidor previamente testado).

2. Teste principal:

- a) Envio de requisições na taxa definida durante 10 segundos.
- b) Pausa de 60 segundos.
- c) Coleta de lixo (*garbage collection*) manual na máquina servidor.
- d) Pausa de 20 segundos.
- e) Aumento da taxa em 50 requisições por segundo.
- f) Repetição dos passos acima até que o servidor comece a apresentar erros de conexão.

3. Pausa de 60 segundos.

4. Repetição dos passos 1 e 2 para o outro mecanismo de threads, em uma nova instância da JVM.

O teste foi finalizado quando o servidor apresentou falhas em três execuções consecutivas, indicando saturação consistente do sistema.

Este teste também foi repetido 45 vezes para cada mecanismo. A duração variou conforme o ponto de saturação: em média, 1 hora e 48 minutos para threads virtuais e 1 hora e 39 minutos para threads tradicionais. O tempo total de experimentação foi de aproximadamente 155 horas.

4.5 Controles Experimentais

Para garantir a validade dos resultados, foram implementados os controles:

1. Garbage Collection Manual: Executado antes de cada teste para minimizar o impacto de coletas automáticas durante as medições. O protocolo incluiu aguardar 60 segundos após qualquer requisição antes da coleta, seguido de 20 segundos de estabilização.
2. Alternância de Ordem: A sequência de execução alternava entre threads tradicionais e virtuais para evitar que efeitos de aquecimento da JVM ou mudanças graduais no sistema favorecessem um mecanismo.
3. Automação: Scripts automatizados padronizaram tempos de warmup, pausas e coleta de lixo.
4. Isolamento de Recursos: Ambas as máquinas foram dedicadas exclusivamente aos experimentos, sem outras aplicações ou serviços em execução.
5. Replicabilidade: Todos os scripts e código-fonte estão disponíveis no repositório GitHub: https://github.com/StephanyeCunto/TCC_Java_Thread_Benchmark

4.6 Métricas Coletadas

As métricas foram coletadas em dois pontos: na máquina servidora e na máquina cliente.

No servidor as métricas foram coletadas através do Java Flight Recorder (JFR), processadas com Java Mission Control (JMC):

1. Utilização média de CPU (%)
2. Consumo de memória RAM (MB)
3. Utilização de heap da JVM (MB)

Os dados foram salvos em arquivos ‘Monitor.jfr’ para cada execução, permitindo análise do comportamento interno da aplicação.

As métricas foram coletadas no cliente pela ferramenta Vegeta durante o envio de requisições:

1. Latência: mínima, máxima, média e percentis (P50, P90, P95, P99)
2. Taxa de transferência (throughput)
3. Taxa de amostragem
4. Taxa de sucesso (%)
5. Volume de dados (bytes de entrada e saída)

Esses dados foram exportados para arquivos JSON para análise e comparação entre os mecanismos de threads.

4.7 Tratamento de erros

Durante os experimentos, foram observados erros de conexão e tempo limite, todos os erros ocorreram devido à falta de resposta do servidor, indicando que o mesmo não conseguiu processar todas as requisições recebidas. Esses erros foram identificados através de tokens gerado pelo Vegeta.

Como exemplo, alguns erros observados:

- EOF (`Client.Timeout exceeded while awaiting headers`)
- `http: server closed idle connection (Client.Timeout exceeded while awaiting headers)`
- `context deadline exceeded (Client.Timeout exceeded while awaiting headers)`

Execuções que apresentaram erros foram mantidas, pois a presença de falhas indica o ponto de saturação do sistema, uma métrica relevante para avaliar escalabilidade.

4.8 Análise Estatística

Para avaliar a necessidade de repetições dos testes foi conduzida uma análise estatística baseada na execução de múltiplas repetições dos experimentos e no calculo do intervalo de confiança (IC) das métricas principais, como latência média, utilização de CPU e consumo de memória. Antes de calcular o intervalo de confiança foram removidos valores discrepantes (*outliers*) utilizando o método do intervalo interquartil (IQR). Isso ajudou

a reduzir a influência de variações extremas e obter estimativas mais representativas do comportamento do sistema.

Os intervalos de confiança foram calculados utilizando a distribuição t de Student, conforme a Equação 4.1:

$$IC = \bar{x} \pm t_{\alpha/2, gl} \cdot \frac{s}{\sqrt{n}} = \bar{x} \pm \text{Margem de Erro} \quad (4.1)$$

Onde:

- \bar{x} é a média amostral da métrica analisada;
- $t_{\alpha/2, gl}$ é o valor crítico da distribuição t de Student para um determinado nível de confiança e graus de liberdade;
- s é o desvio padrão amostral;
- n é o tamanho da amostra (número de execuções);
- $gl = n - 1$ representa os graus de liberdade.

Os valores críticos de t foram obtidos a partir de tabelas estatísticas para os níveis de confiança desejados (95% e 99%) e os graus de liberdade correspondentes. Com isso, foi possível calcular a margem de erro e o intervalo de confiança para cada métrica.

Tabela 2 – Intervalos de confiança das métricas principais para o teste de carga constante

Mecanismo	IC	Lat. média (s)	CPU	Mem. (MB)	Heap (MB)
Threads virtuais	95%	[0,12, 0,14]	[0,04, 0,04]	[415,16, 519,46]	[109,85, 173,56]
Threads virtuais	99%	[0,12, 0,15]	[0,04, 0,04]	[397,14, 537,48]	[98,85, 184,56]
Threads trad.	95%	[0,15, 0,23]	[0,08, 0,08]	[421,85, 525,78]	[156,63, 219,25]
Threads trad.	99%	[0,14, 0,24]	[0,08, 0,08]	[403,79, 543,85]	[145,74, 230,14]

Fonte: Elaboração própria.

A Tabela 2 apresenta os intervalos de confiança calculados para as principais métricas do teste de carga constante. Observa-se que os intervalos são estreitos para latência média e para uso de cpu, indicando baixa variabilidade entre as execuções.

No entanto, os intervalos para consumo de memória e heap são mais amplos, sugerindo maior variação nesses aspectos o que era esperado devido à natureza dinâmica do gerenciamento de memória na JVM.

Esses resultados indicam que o número de execuções adotado foi suficiente para capturar o comportamento médio do sistema de forma estatisticamente confiável, permitindo comparações consistentes entre threads tradicionais e threads virtuais.

4.9 Limitações dos Testes

Como já mencionado, os testes possuem algumas limitações que devem ser consideradas na interpretação dos resultados:

- **Limitação de Hardware:** Não foi possível atingir os limites absolutos da máquina servidora, uma vez que o sistema operacional impõe restrições no número de requisições simultâneas. Os testes, portanto, caracterizam o limite do sistema operacional, não o limite de hardware.
- **Rede Wi-Fi:** A utilização de uma conexão Wi-Fi pode ter introduzido variabilidade adicionais de latência e perda de pacotes. Embora essa limitação afete ambos os mecanismos igualmente, testes em ambiente cabeado poderiam revelar diferenças mais sutis.
- **Ambiente Controlado:** Os testes foram realizados em um ambiente controlado com carga sintética e podem não refletir completamente o comportamento em ambientes de produção com cargas e padrões de uso variados.
- **Simplificação da Aplicação:** o servidor implementado utiliza apenas operações de espera (sleep), não incluindo operações I/O reais como acesso a banco de dados ou chamadas a serviços externos, o que pode limitar a generalização dos resultados

5 Aplicação / Implementação / Experimento

Descrição passo a passo da execução do estudo. Apresentação de scripts, fluxogramas, diagramas ou imagens ilustrativas

6 Conclusão

A conclusão tem o papel de encerrar o trabalho, retomando de forma resumida o que foi feito e destacando as contribuições obtidas. Ela deve responder à pergunta central da pesquisa e deixar claro o que foi aprendido, comprovado ou desenvolvido.

A conclusão deve fechar o trabalho com chave de ouro, respondendo à pergunta de pesquisa, destacando o que foi aprendido e mostrando como o estudo contribui para a área, além de abrir portas para novas pesquisas.

- Relembre brevemente o objetivo geral do estudo e confirme se ele foi atingido.
- Destaque os resultados mais importantes, sem repetir tabelas ou gráficos.
- Foque no que é mais relevante para responder à questão de pesquisa.
- Interprete brevemente o que os resultados significam no contexto do problema.
- Falar das limitações do estudo é opcional, mas recomendado. Reconheça possíveis limitações que possam ter influenciado os resultados.
- Sugestões para trabalhos futuros. Indique possíveis melhorias ou novas abordagens que podem ser exploradas.

Não introduza informações novas que não tenham aparecido no desenvolvimento. Use tempo passado para descrever o que foi feito.

Referências

- AGUAS, D. J. d. O. *Avaliação dos mecanismos de concorrência na API do Java 8*. 2015. Dissertação de Mestrado Instituto Superior de Engenharia do Porto (ISEP). Disponível em: <https://www.proquest.com/openview/dfad25d71a865e5f8f207800736152ed/1?pq-origsite=gscholar&cbl=2026366&diss=y>. Acesso em: 6 nov. 2025.
- BUGHUNT. *A tríade CIA: Confidencialidade, Integridade e Disponibilidade*. 2023. Blog Bughunt. Disponível em: <https://blog.bughunt.com.br/triade-cia/>. Acesso em: 2 jul. 2025.
- CTIR Gov. *CTIR Gov em números*. 2025. Centro de Tratamento de Incidentes de Segurança da Administração Pública Federal. Disponível em: <https://www.gov.br/ctir/pt-br/assuntos/ctir-gov-em-numeros>. Acesso em: 2 jul. 2025.
- GitHub. *State of the Octoverse 2024*. 2024. GitHub Blog. Disponível em: <https://github.blog/news-insights/octoverse/octoverse-2024>. Acesso em: 21 out. 2025.
- GUSTAFSSON, E.; PERSSON, O. N. *Comparison of Concurrency Technologies in Java*. 2024. Masters Thesis Lund University, Department of Computer Science. Disponível em: <https://lup.lub.lu.se/luur/download?func=downloadFile&recordId=9166685&fileId=9166687>. Acesso em: 6 nov. 2025.
- Oracle Corporation. *Virtual Threads*. 2023. Oracle Documentation. Disponível em: <https://docs.oracle.com/en/java/javase/21/core/virtual-threads.html>. Acesso em: 27 out. 2025.
- PANDITA, V. *Benchmarking the Performance of Java Virtual Threads in High-Throughput Workloads*. 2024. Dissertação de Mestrado National College of Ireland, School of Computing. Disponível em: <https://norma.ncirl.ie/8134/>. Acesso em: 6 nov. 2025.
- PECB. *ISO/IEC 27002:2022 Information Security, Cybersecurity and Privacy Protection*. [S.l.], 2022. Disponível em: <https://pecb.com/whitepaper/isoiec-270022022--information-security-cybersecurity-and-privacy-protection>. Acesso em: 2 jul. 2025.
- SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Operating System Concepts*. 9. ed. Hoboken, NJ: John Wiley & Sons, 2018. ISBN 978-1-118-06333-0.
- SOUTO, H. F. d. *Uma análise comparativa entre threads e green threads no Java*. 2024. Artigo Curso de Ciência da Computação, Universidade Federal de Campina Grande (UFCG). Disponível em: <https://dspace.sti.ufcg.edu.br/bitstream/riufcg/38147/1/HIARLY%20FERNANDES%20DE%20SOUTO-ARTIGO-CI%C3%84NCIA%20DA%20COMPUTA%C3%87%C3%83O-CEEI%20%282024%29.pdf>. Acesso em: 6 nov. 2025.
- Stack Overflow. *Stack Overflow Developer Survey 2025: Technology*. 2025. Stack Overflow Blog. Disponível em: <https://survey.stackoverflow.co/2025/technology/>. Acesso em: 27 dez. 2025.
- VERIZON. *2024 Data Breach Investigations Report*. [S.l.], 2024. Disponível em: <https://www.verizon.com/business/resources/Ta53/reports/2024-dbir-data-breach-investigations-report.pdf>. Acesso em: 2 jul. 2025.