Comparação dos Ambientes de Benchmarking

O trabalho *Benchmarking the Performance of Java Virtual Threads in High-Throughput Workloads* criou um ambiente controlado em **AWS EC2 (t3.xlarge)**, enquanto o trabalho *Comparison of Concurrency Technologies in Java* utilizou **dois MacBooks Pro**, porém ambos realizam testes **I/O-bound** semelhantes de acordo com a tabela.

Table 1: Comparação dos Testes I/O-Bound

Aspecto	**Benchmarking the Performance of Java Virtual Threads in High-Throughput Workloads**	**Comparison of Concurrency Technologies in Java**
Ambiente de Execução	Instância AWS EC2 (t3.xlarge). Sistema operacional Linux baseado em Ubuntu. Ambiente virtualizado e compartilhado. Configuração: 4 vCPUs, 16 GB RAM. Possibilidade de variação de desempenho devido à natureza da nuvem.	Duas máquinas físicas (MacBook Pro 2019). Processador Intel Core i7 2,6 GHz (6 núcleos). 16 GB DDR4 de memória. macOS 14.2.1 (64 bits). Conexão Ethernet direta entre as máquinas para eliminar variação de rede.
Framework e Linguagem	Spring Boot 3.2 e Java 21.	Spring Boot 3.2 e Java 21.
Ferramentas Utilizadas	Apache JMeter gerou 100 usuários simultâneos. VisualVM monitorou CPU, memória e threads.	Vegeta responsável por enviar ataques HTTP controlados e medir throughput e latência. VisualVM e coleta da JVM monitoraram CPU, heap, threads vivas, sincronização e pontos de coleta do Garbage Collector.
Arquitetura do Teste	Duas aplicações: Uma que recebe a requisição e dorme 100 ms (simulação de bloqueio I/O). Outra que faz requisições simultâneas para a primeira, sendo testada com threads tradicionais e virtuais.	Duas máquinas físicas interligadas por conexão Ethernet direta. Uma máquina atuou como cliente (gerando carga via Vegeta) e a outra como servidor (executando a aplicação Java). Foram testadas três tecnologias de concorrência: Platform Threads, Virtual Threads e Reactive Streams.
Configuração de Carga	Ramp-up de 60 segundos: o JMeter começou com poucos usuários e foi aumentando gradualmente até chegar a 100 usuários simultâneos. Após o ramp-up, iniciou-se o período de 10 minutos de requisições constantes. Cada usuário fazia requisições GET para o endpoint bloqueante, mantendo o servidor sob rajadas contínuas.	Foram realizados dois tipos de experimentos: (1) Aumento gradual de carga (*load ramping*) para detectar o ponto de saturação. (2) Carga constante para medir estabilidade sob uso intenso. Cada execução foi repetida 5 vezes, descartando outliers e observando CPU, heap e latência.
Comportamento Observado	Platform Threads: alta ocupação de memória e context-switching frequente, resultando em tempo ocioso e menor throughput. Virtual Threads: mantiveram uso de CPU mais estável e melhor latência.	Platform Threads apresentaram maior consumo de memória e limitação de escalabilidade. Virtual Threads demonstraram melhor aproveitamento da CPU e estabilidade sob carga sustentada. Reactive Streams tiveram boa eficiência, mas exigiram maior complexidade de implementação.
Execuções	3 execuções.	5 execuções.