Sumário

1 Lista de Artigos

Nº	Título	Ano	Links	Sistema
1	Avaliação dos mecanismos de concorrência na API do Java 8	2015	Acesso	Não citado
2	Uma análise comparativa entre Threads e Green Threads no Java	2024	Acesso	Ubuntu
3	Comparison of Concurrency Technologies in Java	2024	Acesso	MacOS

2 Anotações

2.1 Avaliação dos mecanismos de concorrência na API do Java 8

Autor(es): DANIEL JORGE DE OLIVEIRA AGUAS
Ano de publicação: 2015

2.1.1 Anotações

- API do Java 8 (Threads, Threadpools, ExecutorService, CountdownLach, ExecutorCompletionService e ForkJoinPools)
- Utilizou os algoritmos de ordenação: Quicksort, Mergesort e Pidgeonholesort com 1000, 5000, 10000, 50000 e 10000 algarismos
- Máquina virtual java utilizada: JVM Hotspot
- Itens que se procurou diferenciar nas máquinas testadas:
  - Quantidade de núcleos físicos
  - Dimensão da cache
  - Presença ou não de tecnologias como o HyperThreading e Turbo Boost nos processadores
- Fatores ignorados devido à dificuldade de obtenção de grandes quantidades de máquinas:
  - Diferentes Sistemas Operativos e diferentes versões deles
- Máquinas utilizados :
  - Intel® Core™ T4300 – 2,1 GHz, 2 núcleos, 1 MB, Nenhuma, 4gb de ram
  - Intel® Core™ i5-4260U – 1,4 GHz, 2 núcleos, 3 MB, Hyper-Threading, Turbo Boost (2.8GHz), 4gb de ram
  - Intel® Core™ i7-3610QM – 2,3 GHz, 4 núcleos, 6 MB, Hyper-Threading, Turbo Boost (2.8GHz), 6gb de ram
- Devido ao consumo de memória, foi necessário aumentar a heap da JVM para 4000 MB. A heap é a área de memória onde os objetos criados em tempo de execução são armazenados.O garbage collector monitora esses objetos e libera memória removendo aqueles que não são mais utilizados, evitando que a aplicação fique sem memória.

- Para produzir o ambiente desejado foram efetuadas algumas alterações na ferramenta de análise JProfiler.
- Rentável, se a duração base de execução do algoritmo permitir que o overhead de tempo com a criação de paralelismo seja compensado, e que o impacto no consumo de memória depende da estrutura do algoritmo.
- Trabalhos futuros:
  - Executar testes em máquinas com processadores que possuam maior capacidade de threads, incluindo processadores com quatro núcleos físicos, para comparar resultados em ambientes com e sem Hyper-Threading.
  - Realizar o estudo em um sistema operacional que facilite o uso de affinity, permitindo maior dedicação do CPU aos testes.
  - Implementar e avaliar outras ferramentas da API do Java 8, ampliando as conclusões sobre programação concorrente.

2.2 Uma análise comparativa entre Threads e Green Threads no Java

Autor(es): Hiarly Fernandes de Souto, Thiago Emmanuel Pereira
Ano de publicação: 2024

2.2.1 Anotações

- Green Threads, chamada de Virtual Threads, design de threads no qual o escalonamento é feito pela própria aplicação por meio de uma biblioteca ou framework.
- Testes de Criação, Inicialização, Junção e Troca de Contexto.
- Threads nativas são eficazes em termos de paralelismo real, elas também são caras em termos de recursos e podem ser difíceis de usar em algumas situações de programação.
- Virtual Threads são mais leves que as threads do kernel em termos de uso de memória, e a sobrecarga de troca de contexto entre elas é muito baixa. Milhões de Virtual Threads podem ser criadas em uma única instância de JVM.
- Máquina utilizado: AMD® Ryzen 7 3700u, 20GB de memória RAM, disco com capacidade de 256GB
- Criou-se 100.000 threads e mediu–se o tempo de execução para a criação desses objetos

- Iniciou-se um total de 10.000 threads que não realizavam operações específicas
- Cada thread aguarda por 100 milissegundos, invoca o método Thread.yield() (Mudar Contexto)
- Instanciar Threads, Iniciar Threads,Join Thread, Mudar Contexto a Thread Virtual é mais rápido
- Ausência de diversidade nas configurações de infraestrutura
- A realização dos testes do estudo em ambientes distintos poderia levar a resultados divergentes, uma vez que a JVM (Java Virtual Machine) poderia realizar otimizações específicas de acordo com o hardware da máquina utilizada

2.3 Comparison of Concurrency Technologies in Java

Autor(es): Elias Gustafsson, Oliver Nederlund Persson
Ano de publicação: 2024

2.3.1 Anotações

- Foram realizadas análises de pilha, tráfego de rede e impacto de ferramentas de profiling.
- Reactive Systems são sistemas assíncronos, escaláveis, resilientes e responsivos, que usam fluxos de dados e mensagens para lidar com alta carga e eventos em tempo real, garantindo desempenho mesmo diante de falhas.
- Pergunta 1: Quais são as diferenças entre essas diferentes técnicas de simultaneidade dentro da JVM em sistemas de alta carga?
- Pergunta 2: Threads virtuais são uma alternativa viável para substituir threads de plataforma e fluxos de dados assíncronos em aplicações de alta carga?
- Maquina utlizada: (MacBook Pro 2019) 2.6 GHz-6 core Intel Core i7, Memory 16 GB 2667 MHz DDR4, MacOS 14.2.1.
- Desafios em Java:
  - Garbage Collector (GC) — coleta de lixo pode causar pausas variáveis.
  - JRE e JVM — diferentes máquinas virtuais podem apresentar comportamentos distintos.

  - Just-In-Time (JIT) Compilation — otimizações dinâmicas podem alterar resultados entre execuções.

- Três servidores foram implementados utilizando threads da plataforma, threads virtuais e programação reativa.
- Testes com taxa de requisições constante foram conduzidos para avaliar a estabilidade.