Processando o Saber
eISSN 2179-5150
FATEC Praia Grande · fatecpg.edu.br/revista
submetido: Jan/2025 · **aceito**: Mar/2025 · **publicado**: Jun/2025
DOI 10.5281/zenodo.15477083

Análise comparativa de desempenho das funções *lambda* entre as linguagens Go e Java

Comparative performance analysis of lambda functions between go and java languages

Jônatas C. Dias
Fatec Praia Grande
jonatas.dias2@fatec.sp.gov.br

Cassio M. Izidoro
Fatec Praia Grande
cassiomartinez2@gmail.com

Jeferson Cerqueira Dias
Fatec Itaquera
jefersoncdias@hotmail.com

RESUMO

À medida que a geração de dados cresce exponencialmente, as empresas deparam-se com desafios cada vez maiores no que diz respeito ao armazenamento, processamento e análise eficiente dessas informações. A adoção de funções *Lambda* tem-se mostrado uma solução viável e econômica para lidar com o grande volume de dados. Nesse contexto, a comparação de desempenho entre as linguagens de programação *Go* e *Java* é de interesse para desenvolvedores e empresas em busca de melhor performace de suas funções *Lambda*. Este estudo tem como objetivo analisar e comparar o desempenho dessas funções nessas duas linguagens e fornecer informações relevantes para a seleção da linguagem mais adequada às necessidades e objetivos de desempenho. A metodologia empregada envolve a implementação de diferentes abordagens para a série de *Fibonacci* em ambas as linguagens, com medição do tempo de execução em nanosegundos. Os resultados indicam um melhor desempenho da linguagem *Go* em relação a linguagem *Java*, oferecendo *insights* para os desenvolvedores na escolha da linguagem mais adequada. Além disso, dada a crescente demanda por aplicações em nuvem, é crucial aprofundar o conhecimento sobre computação em nuvem e o uso de tecnologias relacionadas para lidar eficientemente com grandes volumes de dados.

PALAVRAS-CHAVE: Funções *Lambda*, *Go*, *Java*, Desempenho, Computação em nuvem.

ABSTRACT

As data generation grows exponentially, companies face increasing challenges in storage, processing, and efficient analysis of this information. The adoption of Lambda functions has proven to be a viable and cost-effective solution for handling large volumes of data. In this context, comparing the performance between the Go and Java programming languages is of interest to developers and companies seeking improved performance of their Lambda functions. This study aims to analyze and compare the performance of these functions in both languages and provide relevant information for selecting the most suitable language based on performance needs and goals. The methodology employed involves implementing different approaches for the Fibonacci series in both languages, with measurement of execution time in nanoseconds. The results indicate better performance of the Go language compared to Java, offering insights for developers in choosing the most suitable language. Furthermore, given the growing demand for cloud applications, it is crucial to deepen knowledge about cloud computing and the use of related technologies to efficiently handle large volumes of data.

KEY-WORDS*: Lambda functions, Go, Java, Performance, Cloud computing.*

INTRODUÇÃO

Com o aumento contínuo da geração de dados, as empresas estão lidando com volumes cada vez maiores de informações, o que pode resultar em grandes desafios em relação ao armazenamento, processamento e análise de dados. De acordo com a *International Data Corporation* (IDC), a previsão é que a quantidade de dados gerados em todo o mundo alcance 175 zettabytes[1] até 2025 (EYUPOGLU, 2020; JANEV, 2021). Com esse grande volume de dados, as empresas precisam lidar com uma série de desafios, como o custo de armazenamento e processamento, bem como a necessidade de manter uma infraestrutura robusta e escalável para gerenciar esses dados. Nesse sentido, a computação em nuvem tem se mostrado uma solução viável e econômica para o processamento de *Big Data*, com um crescimento de 24,1% no mercado global de serviços em nuvem em 2020, totalizando US$ 312 bilhões em investimentos, de acordo com a (EYUPOGLU, 2020; JANEV, 2021).

[1] Zettabyte é uma unidade de medida de armazenamento de dados que equivale a 1 trilhão de gigabytes, pode-se dizer que seria necessário cerca de 250 bilhões de DVDs para armazenar essa quantidade de informações.

Com o advento dos serviços em nuvem, as funções *Lambda*[2] se tornaram uma abordagem cada vez mais popular para a implementação de lógicas do lado do servidor sem a necessidade de gerenciamento de infraestrutura. As funções *Lambda* são uma abordagem *serverless*[3] de computação em nuvem que permite aos desenvolvedores executarem código sem a necessidade de gerenciar servidores ou ambientes de execução.

A comparação de desempenho entre as linguagens de programação Go e Java pode fornecer *insights* valiosos para desenvolvedores e empresas que buscam alto desempenho em suas aplicações em nuvem. Diante disso, é fundamental que os profissionais da área de tecnologia estejam constantemente atualizados e em busca de soluções inovadoras para lidar com o grande volume de dados e se manterem competitivos no mercado.

A Linguagem Go (LG) se posiciona como a 16ª linguagem de programação mais popular enquanto a Linguagem Java (LJ) como a segunda mais popular. Além disso, LG é conhecida por sua eficiência e desempenho em sistemas distribuídos, enquanto LJ é amplamente utilizado em grandes empresas e projetos de grande escala (O'GRADY, 2022).

De acordo com a pesquisa realizada pela empresa *SlashData – Developer Nation*[4], a LJ é a segunda linguagem de programação mais utilizada no desenvolvimento de aplicações em nuvem, enquanto a LG ocupa a quinta posição. Esses resultados evidenciam que ambas as linguagens são amplamente empregadas em projetos na nuvem (KORAKITIS et al., 2022). Além disso, é importante destacar que o desempenho das funções *Lambda* pode ser afetado por diversos fatores além da linguagem de programação escolhida. A escolha de uma arquitetura adequada para a função *Lambda* e o uso de boas práticas de programação podem ter um impacto significativo no desempenho das funções (AWS, 2014).

Outra pesquisa realizada pela Gartner Inc. revela que a demanda por aplicações em nuvem está aumentando significativamente e tem previsão de superar os gastos com tecnologia da informação tradicional até 2025. De acordo com os dados apresentados, 51% dos gastos em *software* de aplicativos, *software* de infraestrutura, serviços de processos de negócios e infraestrutura de sistemas serão direcionados para a nuvem pública em 2025, em comparação com 41% em 2022. Além disso, a pesquisa indica que quase dois terços dos gastos em *software*

[2] As funções *Lambda* são explicadas em detalhes mais adiante na seção de referencial teórico neste artigo.
[3] Serverless é uma abordagem de desenvolvimento e implantação de aplicações em nuvem que permite aos desenvolvedores concentrarem-se na lógica de negócios, enquanto a infraestrutura de servidores é gerenciada e provisionada pela plataforma de computação em nuvem.
[4] A *SlashData - Developer Nation* é uma comunidade global que envolve milhares de programadores de todas as formas e tamanhos em todo o mundo, permitindo-lhes compararem-se com a nação de programadores. Estamos empenhados em facilitar a contribuição da comunidade e a partilha de conhecimentos, e promover a diversidade e a inclusão no ecossistema desenvolvedor.

de aplicativos serão voltados para tecnologias em nuvem até 2025. Esses resultados sugerem que as empresas devem considerar a adoção de tecnologias em nuvem em suas estratégias de TI para se manterem competitivas e se prepararem para o futuro (GARTNER, 2021).

Diante da crescente demanda por aplicações em nuvem e a necessidade de alta *performance*, é importante avaliar o desempenho das linguagens de programação nas funções *Lambda*. A comparação entre as linguagens Go e Java pode fornecer *insights* valiosos para desenvolvedores e empresas. Mas afinal, qual das linguagens apresenta melhor desempenho nas funções *Lambda*?

Para responder esta questão, este estudo tem como objetivo principal a missão de comparar o desempenho das funções *Lambda* nas linguagens de programação Go e Java. O estudo busca identificar qual linguagem de programação oferece melhor desempenho para funções *Lambda* e fornecer informações valiosas para desenvolvedores sobre a escolha da linguagem mais adequada para suas necessidades e objetivos de desempenho. Isto delineou os objetivos específicos como segue:

a) Compreender o conceito de funções *Lambda* e sua aplicação em linguagens de programação modernas, como Go e Java.
b) Analisar as diferentes abordagens para a implementação da série de *Fibonacci*, incluindo iteração, recursão e memorização.
c) Implementar as funções *Lambda* em Go e Java para calcular a série de *Fibonacci* e medir o tempo de execução usando o tempo de resposta médio (em nanosegundos) para 100 solicitações consecutivas.
d) Comparar e analisar os resultados das medições de desempenho para determinar qual linguagem de programação oferece melhor desempenho para funções *Lambda*.
e) Discutir as implicações dos resultados da análise de desempenho e fornecer recomendações para desenvolvedores escolherem a linguagem de programação apropriada para suas funções *Lambda*, com base em suas necessidades e objetivos de desempenho.

Cabe observar que a escolha da utilização da séria de Fibonacci é vantajosa por vários aspectos, por exemplo:

a) **Simplicidade e Clareza:** A série de Fibonacci é um problema clássico na computação e é amplamente compreendida. Isso torna os resultados dos testes mais fáceis de interpretar e comunicar.
b) **Facilidade de Implementação:** A lógica para calcular a série de Fibonacci pode ser implementada de forma relativamente simples e rápida em diferentes linguagens de programação.

c) **Reprodutibilidade:** Os resultados dos testes utilizando a série de Fibonacci são facilmente reproduzíveis, o que é importante para garantir a confiabilidade dos resultados.

A contribuição deste artigo está presente no avanço do conhecimento sobre o desempenho de funções *Lambda* em linguagens de programação modernas, além de auxiliar os desenvolvedores na escolha da linguagem mais adequada para suas necessidades de desenvolvimento e desempenho. Para tanto, serão realizadas análises comparativas rigorosas, com medidas objetivas de desempenho, a fim de fornecer resultados confiáveis e úteis para a comunidade de desenvolvedores.

1. MATERIAIS E MÉTODOS

Para atingir os objetivos elencados neste estudo, foi estabelecido um teste para a execução da série de *Fibonacci* seguido da medição do tempo de execução. Este segue a orientação de padrões estabelecidas pela *International Organization for Standardization* para a comparação do desempenho das funções *Lambda* entre as linguagens Go e Java para realização do calculo da série de *Fibonacci* (ISO/IEC-17025, 2005). As diferentes abordagens devem incluir as informações conforme apresentadas no Quadro 1.

Quadro 1 – Abordagens consideradas para o teste

ID	Abordagem	Descrição
1	Descrição do procedimento experimental	O procedimento experimental consiste na implementação das funções *Lambda* em Go e Java para calcular a série de *Fibonacci* usando diferentes abordagens, como iteração, recursão e memorização. Para medir o tempo de execução, serão realizadas 100 solicitações consecutivas em cada abordagem.
2	Ambiente de teste	O ambiente de teste utilizado consiste em um computador com processador Intel Core i3, 8 GB de memória RAM e sistema operacional Windows 10. Serão utilizadas as versões mais recentes das linguagens Go e Java para implementar as funções *Lambada*.
3	Métodos de medição	O tempo de execução será medido usando o tempo de resposta médio em nanosegundos para 100 solicitações consecutivas em cada abordagem. Serão utilizadas as ferramentas de medição disponíveis nas linguagens Go e Java para medir o tempo de execução.
4	Cenários de teste	Serão criados 5 cenários de teste, cada um correspondente a um número diferente de elementos da série de Fibonacci, sendo eles 10, 20, 30, 40 e 50. Para cada cenário de teste, serão realizadas 100 solicitações consecutivas em cada abordagem. Os resultados esperados para cada cenário de teste serão calculados previamente e comparados com os resultados obtidos durante o experimento.
5	Coleta e análise de dados	Os dados de desempenho das funções *Lambada* serão coletados e armazenados em tabelas, contendo o tempo de execução médio em nanosegundos para cada abordagem em cada cenário de teste. Os dados serão analisados usando técnicas estatísticas apropriadas para identificar diferenças significativas entre as abordagens em cada cenário de teste.
6	Limitações do estudo	As limitações do estudo incluem as especificidades do ambiente de teste utilizado, bem como as limitações das ferramentas de medição disponíveis nas linguagens Go e Java. Além disso, o estudo será realizado em um único computador, o que pode afetar a replicabilidade dos resultados em outros ambientes de teste
7	Considerações éticas	Este estudo não envolve o uso de dados confidenciais ou pessoais, portanto, não há considerações éticas a serem discutidas.

Fonte: Elaborado pelos Autores

Ao seguir as normas estabelecidas pela ISO, é possível garantir a qualidade e a transparência da seção de Materiais e Métodos, permitindo que os resultados do experimento sejam interpretados de forma clara e objetiva. Por fim, é importante destacar que os resultados obtidos neste estudo podem ser úteis para desenvolvedores de *software*, pesquisadores e acadêmicos interessados em melhorar o desempenho de funções *Lambda* em linguagens de programação. Os resultados também podem ser usados para orientar futuros estudos na área de otimização de funções *Lambda*.

Para evitar viés de implementação das funções *Lambda* em Go e Java[5], pois elas podem variar de acordo com as habilidades e experiência dos programadores, foi então proposto um algoritmo comum de implemetação, padronizando as implementações para minimizar esse viés, conforme a Figura 1.

[5] Diferenças na otimização do código podem afetar os resultados do experimento.

Figura 1 – Algoritmo base de escrita do código

ID	Comando
01	Função Fibonacci(n: inteiro) -> inteiro
02	Se n for menor ou igual a 1 então
03	Retorne n
04	Senão
05	a <- 0
06	b <- 1
07	Para i de 2 até n faça
08	temp <- b
09	b <- a + b
10	a <- temp
11	Fim Para
12	Retorne b
13	Fim Se
14	Fim Função
15	
16	Algoritmo Principal
17	Para cada i de 0 até 10 faça
18	Escreva "Fibonacci(", i, "): ", Fibonacci(i)
19	Fim Para
20	Fim Algoritmo

Fonte: Elaborado pelos Autores

O algoritmo da **Figura 1** segue a mesma lógica das implementações em Go e Java, onde a função Fibonacci calcula o n-ésimo número da série de Fibonacci de forma iterativa, e o algoritmo principal testa a função para os valores de entrada de 0 a 10 e exibe os resultados. Além disto outros cuidados foram tomados, como por exemplo, a igualdade na avaliação do tempo de execução entre as linguagens Go e Java, é importante selecionar ferramentas que sejam amplamente reconhecidas e utilizadas pela comunidade, ofereçam funcionalidades semelhantes e permitam medições precisas e confiáveis. Por esta razão duas ferramentas foram utilizadas conforme estão apresentadas no Quadro 2.

Quadro 2 – Ferramentas de teste de *Benchmark*[6] de ambas as linguagens

ID	Ferramentas	Descrição
1	**GO** *Benchmarking*	A ferramenta de *benchmarking* integrada ao pacote *testing* do Go permite medir o desempenho de funções de maneira sistemática. Isso pode ser feito criando funções de *benchmark* no arquivo _test.go e executando-os usando o comando go test '-bench=.'
2	**JAVA** JMH (Java *Microbenchmark Harness*)	Esta é uma ferramenta avançada para testes de *benchmark* em Java, desenvolvida pelo OpenJDK. O JMH permite criar *benchmarks* precisos e detalhados, lidando com questões como aquecimento da JVM e otimizações de código.

Fonte: Elaborado pelos Autores

Essas ferramentas podem ser utilizadas para medir o tempo de execução das funções *Lambada* em Go e Java de maneira precisa e confiável, ajudando a comparar o desempenho entre as linguagens de forma eficaz.

2. REFERENCIAL BIBLIOGRÁFICO

O referencial bibliográfico abordará as funções *Lambda*, a linguagem Go e a linguagem Java, destacando suas características e utilização. Será discutido o conceito de funções *Lambda*, sua origem na teoria da computação e matemática, e seu papel na programação funcional e na computação em nuvem.

Em relação à linguagem Go, será explorada sua origem, enfatizando suas características. Será discutido como o Go é utilizado no desenvolvimento de *software* escalável, concorrente e em sistemas distribuídos, como na Internet das Coisas (IoT) e em plataformas de nuvem.

Por fim, o texto apresentará a linguagem Java, destacando suas características, portabilidade, segurança e eficiência. Será explicado o conceito de portabilidade do código Java, que pode ser executado em diferentes plataformas sem a necessidade de recompilação. Serão abordadas as características de segurança, como a verificação de tipos em tempo de compilação e o sistema de gerenciamento de memória automatizado. Além disso, serão discutidos aspectos de eficiência, como a compilação *just-in-time* e o gerenciamento otimizado de memória. Por fim, será enfatizada a orientação a objetos do Java, com recursos avançados como herança, polimorfismo e encapsulamento.

[6] Um teste de *benchmark*, também conhecido como teste de desempenho, é um tipo de teste de software projetado para avaliar o desempenho relativo de um sistema, componente ou algoritmo em relação a um conjunto de critérios predefinidos. O objetivo principal de um teste de *benchmark* é medir e comparar o desempenho de diferentes sistemas ou componentes para determinar qual deles oferece melhor desempenho em uma determinada situação ou carga de trabalho.

Ao longo do referencial bibliográfico, serão utilizadas diversas fontes, incluindo artigos científicos, livros e documentação oficial das linguagens e tecnologias abordadas. Essas fontes fornecerão embasamento teórico e prático para a discussão sobre funções *Lambda*, linguagem Go e linguagem Java, contribuindo para uma compreensão abrangente dos temas explorados.

2.1 FUNÇÕES *LAMBDA*

As funções *Lambda*, também conhecidas como funções anônimas, têm uma longa história na teoria da computação e matemática. Elas foram inicialmente descritas por Alonzo Church na década de 1930 em seu trabalho sobre o cálculo *Lambda*, um sistema formal que permite a representação de funções matemáticas de forma precisa e concisa (Barendregt, 1997).

Desde então, as funções *Lambda* se tornaram uma importante ferramenta na programação funcional, uma abordagem de programação que enfatiza a composição de funções e a imutabilidade de dados. As funções *Lambda* permitem a criação de funções simples e flexíveis que podem ser passadas como argumentos para outras funções e retornadas como valores de outras funções (Bird, 1998).

As funções *Lambda* são amplamente utilizadas na computação em nuvem, permitindo a criação de funções escaláveis que respondem a eventos específicos, como o upload de arquivos para um armazenamento em nuvem (EYUPOGLU, 2020; JANEV, 2021). Além disso, a Netflix, empresa líder em *streaming* de vídeo, utiliza a tecnologia de funções *Lambda* para melhorar o desempenho e oferecer uma experiência de visualização de alta qualidade aos seus usuários.

Um exemplo de uso das funções *Lambda* pela Netflix é a geração de recomendações personalizadas de conteúdo com base nos hábitos de visualização dos usuários. Para realizar essa tarefa em tempo real, a Netflix usa funções *Lambda* em sua arquitetura de microsserviços, que processa dados dos usuários e gera sugestões personalizadas de filmes e séries. Essa abordagem permite que a Netflix ofereça um serviço mais eficiente e personalizado aos seus clientes (EYUPOGLU, 2020; JANEV, 2021).

As funções *Lambda* permitem que a Netflix processe grandes volumes de dados em tempo real, sem a necessidade de configurar e gerenciar servidores. Isso permite que eles gerem recomendações personalizadas para seus usuários de forma rápida e eficiente, o que melhora a experiência do usuário e aumenta a satisfação do cliente.

Além disso, a Netflix também usa funções *Lambda* para escalonar automaticamente sua infraestrutura de *streaming* de vídeo em resposta ao aumento da demanda. Isso significa que, quando muitas pessoas estão assistindo simultaneamente, a Netflix pode adicionar automaticamente mais servidores para garantir que todos tenham uma experiência de visualização suave (AWS, 2014).

2.1.1 Função *Lambda* orientada a Evento

As funções *Lambda* têm sido cada vez mais utilizadas na programação orientada a eventos e em sistemas de computação em nuvem. Na programação orientada a eventos, as funções *Lambda* são usadas para criar manipuladores de eventos, que são acionados em resposta a eventos específicos, como o clique de um botão ou o recebimento de um *e-mail*. Na computação em nuvem, as funções *Lambda* são usadas para criar serviços "sem servidor", que executam código em resposta a eventos específicos, como o *upload* de um arquivo ou o recebimento de uma solicitação HTTP. Isso permite a criação de sistemas escaláveis e eficientes, sem a necessidade de gerenciar servidores ou infraestrutura complexa (PATTERSON, 2019).

As funções *Lambda* são frequentemente usadas em conjunto com outros serviços em nuvem, como o *Amazon S3*, o *Amazon API Gateway e o Amazon DynamoDB*. Por exemplo, um desenvolvedor pode usar uma função *Lambda* para processar arquivos enviados para o Amazon S3, executar validações de entrada em uma solicitação HTTP recebida pelo *Amazon API Gateway* ou realizar cálculos em tempo real em dados armazenados no *Amazon DynamoDB*. Isso permite a criação de aplicativos escaláveis e resilientes que podem ser facilmente integrados com outros serviços em nuvem para fornecer recursos adicionais, como segurança, monitoramento e gerenciamento de dados (AWS, 2023; PATTERSON, 2019).

2.2 LINGUAGEM GO

A linguagem de programação Go é uma ferramenta de código aberto criada pelo Google em 2007, que ganhou popularidade em razão de suas características de eficiência, segurança e simplicidade. Também conhecida como *Golang*, essa linguagem foi desenvolvida para simplificar a criação de *software* escalável e concorrente em sistemas distribuídos, como os usados em aplicativos de Internet das Coisas (IoT) e em grandes plataformas de nuvem. A alta performance do Go tem atraído a atenção de desenvolvedores em todo o mundo, tornando-a uma das linguagens mais utilizadas atualmente.

2.2.1 Características da Linguagem Go

Segundo Donovan e Kernighan (2016), uma das principais características do Go é sua simplicidade, que facilita a leitura e a manutenção do código. Essa simplicidade é resultado de uma sintaxe clara e de um conjunto de recursos limitados, mas eficazes. A linguagem apresenta um pequeno número de palavras-chave e uma biblioteca padrão que cobre as principais funcionalidades necessárias ao desenvolvimento de aplicativos. Além disso, o Go é uma linguagem tipada estaticamente, o que significa que os tipos de dados são definidos em tempo de compilação, reduzindo a chance de erros durante a execução. Essas características fazem do Go uma opção atraente para projetos que exigem simplicidade e confiabilidade.

A concorrência é uma característica fundamental da linguagem Go. O modelo de concorrência baseado em *goroutines*[7] e canais oferece uma maneira elegante e eficiente de lidar com tarefas assíncronas em programas distribuídos e de alta concorrência. As *goroutines* são threads leves que são executadas em tempo de execução e podem ser criadas facilmente usando a palavra-chave "go". Os canais permitem que as *goroutines* comuniquem e coordenem suas atividades de maneira segura e eficiente. A combinação de *goroutines* e canais é uma maneira poderosa de escrever programas concorrentes de alta performance e escaláveis (DONOVAN; KERNIGHAN, 2015).

[7] Goroutines são uma forma leve e eficiente de concorrência em Go. "Goroutines são execuções leves, concorrentes, que são gerenciadas pela máquina virtual Go" (GO,2023).

A compilação rápida é outra característica importante da linguagem Go. A equipe de desenvolvimento do Go projetou a linguagem com a intenção de tornar a compilação rápida, permitindo que os desenvolvedores possam testar e iterar rapidamente no código. A maioria dos programas Go compila em questão de segundos, mesmo para projetos grandes e complexos. (DONOVAN & KERNIGHAN, 2015).

2.2.2 Utilização da Linguagem Go

É importante destacar que a linguagem também tem sido utilizada em projetos para dispositivos móveis. A partir de 2015, o Google começou a oferecer suporte oficial para a criação de aplicativos para Android usando o Go, por meio da biblioteca Go Mobile. Além disso, a empresa também lançou o projeto *Fuchsia*, um sistema operacional em desenvolvimento que utiliza o Go como uma das principais linguagens de programação. Esse suporte ao Go em diferentes plataformas tem contribuído para a crescente popularidade da linguagem e para sua adoção em diversos projetos de desenvolvimento JetBrains (2021).

Devido à natureza distribuída dos sistemas IoT, é importante que a linguagem utilizada seja capaz de se adaptar a diferentes plataformas e dispositivos. O Go tem sido cada vez mais utilizado em projetos IoT, com exemplos como o projeto *TinyGo*, que permite a compilação de código Go para microcontroladores e dispositivos embarcados (TINYGO, 2023). Além disso, o suporte nativo do Go para concorrência e comunicação entre processos torna a linguagem uma opção interessante para o desenvolvimento de aplicativos IoT que precisam lidar com múltiplas tarefas simultaneamente.

Embora a linguagem Go seja frequentemente descrita como uma linguagem de programação procedural, ela também oferece suporte à programação orientada a objetos (POO). Embora a abordagem da linguagem à POO seja diferente de outras linguagens como Java e C++, ela permite a criação de tipos estruturados e a definição de métodos para esses tipos. Diferentemente de outras linguagens de programação orientadas a objetos, como Java e C++, Go não possui classes. Em vez disso, Go usa a ideia de tipos definidos pelo usuário para representar objetos e métodos que agem sobre esses tipos. A definição de um tipo pode incluir campos de dados e métodos associados a esse tipo (DONOVAN; KERNIGHAN, 2015).

Os métodos definidos em um tipo em Go são semelhantes a funções, mas com uma diferença importante. Em vez de ter um parâmetro explícito para o objeto, o objeto é passado como um receptor no início da lista de argumentos. Isso permite que os métodos sejam definidos

em tipos definidos pelo usuário e chamados usando a sintaxe de ponto, semelhante ao que seria feito em outras linguagens orientadas a objetos.

De acordo com a documentação oficial da linguagem Go não é uma linguagem orientada a objetos, porem, oferece suporte a conceitos orientados a objetos, como por exemplo tipos e métodos associados. Isso significa que, embora a abordagem da linguagem Go à POO seja diferente da maioria das linguagens orientadas a objetos, ainda é possível criar tipos estruturados e definir métodos para esses tipos. A documentação do Go fornece informações detalhadas sobre como usar esses recursos de POO na linguagem (DONOVAN; KERNIGHAN, 2015).

2.2.3 Funções *Lambda* em Go

De acordo com a documentação oficial da linguagem Go, uma das principais vantagens da linguagem em funções *Lambda* é sua eficiência e baixo consumo de recursos. Isso é especialmente importante em ambientes de computação serverless, onde o tempo de execução e o uso de recursos podem afetar significativamente o desempenho da aplicação. A linguagem Go é conhecida por ter um tempo de execução rápido e uma baixa sobrecarga, o que significa que as funções *Lambda* implementadas em Go podem ser executadas rapidamente com um mínimo de recursos (DONOVAN; KERNIGHAN, 2015).

Uma das principais características que torna o Go uma excelente opção para funções *Lambda* é sua escalabilidade. Como mencionado anteriormente, a linguagem foi projetada para ser eficiente e escalável, o que a torna uma escolha ideal para aplicações que precisam lidar com um grande volume de requisições. Em particular, o Go oferece suporte à programação concorrente, que permite a execução paralela de código em várias *goroutines*. Isso possibilita que as aplicações possam escalar rapidamente de acordo com a demanda, o que é especialmente importante em ambientes de computação sem servidor como as funções *Lambda* (PATTERSON, 2019).

Em resumo, o Go é uma linguagem de programação de código aberto desenvolvida pelo Google, que se destaca por sua eficiência, simplicidade, segurança e capacidade de programação concorrente. Essas características tornam o Go uma opção popular para desenvolvimento de *software* em sistemas distribuídos, IoT e em grandes plataformas de nuvem. A simplicidade do Go facilita a leitura e manutenção do código, enquanto a eficiência de desempenho, a segurança e a portabilidade tornam.

2.3 LINGUAGEM JAVA

Java é uma linguagem de programação de alto nível, orientada a objetos e criada em 1995 pela Sun Microsystems, atualmente propriedade da Oracle Corporation. É uma das linguagens de programação mais populares e utilizadas no mundo, devido às suas características de portabilidade, segurança, eficiência e ampla disponibilidade de bibliotecas e *frameworks*.

2.3.1 Características da Linguagem Java

A portabilidade é uma das principais vantagens da linguagem de programação Java. Essa característica permite que o código escrito em Java seja executado em qualquer plataforma que possua uma JVM instalada, sem a necessidade de realizar a recompilação do código. Com isso, a linguagem se torna uma escolha comum para desenvolvimento de aplicativos em diferentes arquiteturas e sistemas operacionais (KLEMS, 2018; MUGHAL; RASMUSSEN, 2016).

Outra característica importante do Java é a sua segurança. A linguagem possui um sistema de segurança robusto, que inclui uma verificação de tipos de dados em tempo de compilação e um sistema de gerenciamento de memória automatizado que ajuda a evitar vulnerabilidades comuns como *buffer overflows* e vazamentos de memória. Além disso, o Java possui uma série de recursos de segurança, como a capacidade de executar programas em um ambiente sandbox e a utilização de criptografia para garantir a privacidade dos dados (KLEMS, 2018; MUGHAL; RASMUSSEN, 2016).

A eficiência é outra característica fundamental da linguagem Java. Embora seja uma linguagem interpretada, a JVM utiliza a técnica de compilação *just-in-time* (JIT), que compila o código em tempo de execução, melhorando significativamente a velocidade de execução do código. Além disso, a JVM possui um sistema de gerenciamento de memória eficiente que permite a alocação e liberação de memória de forma otimizada, melhorando a performance das aplicações Java (KLEMS, 2018; MUGHAL; RASMUSSEN, 2016).

Java é uma linguagem orientada a objetos, o que significa que ela se concentra na definição de objetos e na interação entre eles. Isso permite uma maior modularização do código e uma abstração mais clara dos conceitos, facilitando a manutenção e a reutilização do código. A linguagem Java também possui recursos avançados de orientação a objetos, como herança, polimorfismo e encapsulamento, que permitem a criação de hierarquias de classes e a reutilização de código de maneira eficiente (KLEMS, 2018; MUGHAL; RASMUSSEN, 2016).

A ampla disponibilidade de bibliotecas e *frameworks* é outra característica importante da linguagem Java. Existem inúmeras bibliotecas e *frameworks* disponíveis para a linguagem, que podem ser utilizados para facilitar o desenvolvimento de diferentes tipos de aplicações, como aplicações *web*, *mobile*, de *desktop*, entre outras. Alguns exemplos de *frameworks* populares em Java são o *Spring*[8], o *Hibernate*[9], o *Struts*[10] e o JSF[11] (KLEMS, 2018; MUGHAL; RASMUSSEN, 2016).

2.3.2 Utilização da Linguagem Java

No mundo empresarial, o Java é amplamente utilizado para o desenvolvimento de *software* de missão crítica, como sistemas bancários, sistemas de gerenciamento de estoque e sistemas de gerenciamento de recursos humanos. Isso ocorre porque o Java é capaz de lidar com grandes quantidades de dados e é uma escolha segura e confiável para sistemas empresariais.

Além disso, o Java é amplamente utilizado no desenvolvimento de aplicativos móveis. Por meio do *Android Studio*, os desenvolvedores podem criar aplicativos móveis para dispositivos Android usando a linguagem Java, o que é importante, uma vez que o sistema operacional Android é o mais utilizado no mundo (DIETZ, 2018). No entanto, para o desenvolvimento de aplicativos para o sistema operacional *iOS* da *Apple*, o uso de uma ferramenta adicional, como o *Xamarin* ou o *Cordova*, é necessário, uma vez que o Java não é uma linguagem nativa do iOS (FARRELL, 2022).

Outra área em que o Java é amplamente utilizado é no desenvolvimento de aplicativos de servidor. O Java é frequentemente usado como a linguagem de programação principal para servidores da *web* e aplicativos de servidor em geral. Isso se deve à sua capacidade de lidar com múltiplas conexões de clientes simultaneamente e à sua eficiência em lidar com grandes quantidades de dados. Além disso, o Java é uma linguagem altamente modular, o que significa que é fácil de manter e atualizar (GEARY, 2016).

[8] *Spring* – É um *framework* de desenvolvimento de aplicações Java que facilita a criação de aplicativos corporativos robustos e escaláveis (VMware, 2023).
[9] *Hibernate* – É um *framework* de mapeamento objeto-relacional para Java que simplifica o acesso a bancos de dados, proporcionando persistência transparente (REDHATE, 2023).
[10] Struts – É um *framework* de desenvolvimento web em Java, que facilita a criação de aplicativos escaláveis e modulares (FOUNDATION, 2023).
[11] JSF – O JavaServer Faces (JSF) é um *framework* de desenvolvimento web em Java que simplifica a criação de interfaces de usuário interativas. Ele é baseado em componentes e eventos (DEVMEDIA, 2023).

2.3.3 Funções *Lambda* em Java

As funções *Lambda* em Java foram adicionadas na versão 8 da linguagem como uma forma de trazer a programação funcional para a plataforma Java. De acordo com a documentação oficial da Oracle, uma expressão *Lambda* é uma função anônima para criar instâncias de uma interface funcional. Com uma expressão *Lambda*, pode-se escrever uma função que pode ser passada como argumento, assim como se faz com um objeto. (KLEMS, 2018; MUGHAL; RASMUSSEN, 2016).

Isso significa que funções *Lambda* em Java permitem que o programador defina uma função concisa e anônima que possa ser usada para representar um bloco de código. A sintaxe básica para definir uma função *Lambda* envolve a definição dos parâmetros entre parênteses, seguido de uma seta "->" e o corpo da função (KLEMS, 2018; MUGHAL; RASMUSSEN, 2016).

A documentação oficial do Java também destaca a importância das interfaces funcionais na utilização de funções *Lambda*. Uma interface funcional é uma interface que possui apenas um método abstrato e, portanto, pode ser usada como um tipo de referência para uma função *Lambda* (KLEMS, 2018; MUGHAL; RASMUSSEN, 2016).

As funções *Lambda* em Java oferecem uma série de vantagens importantes para a programação moderna. De acordo com a documentação oficial da Oracle, uma das principais vantagens é a facilidade de escrever código mais legível e conciso. As funções *Lambda*s também podem ser usadas para escrever código mais modular e extensível, permitindo que as funções sejam passadas como argumentos para outros métodos e fornecendo maior flexibilidade na implementação do código (KLEMS, 2018; MUGHAL; RASMUSSEN, 2016).

Em resumo, a linguagem de programação Java possui diversas características que a tornam uma escolha popular e viável para o desenvolvimento de aplicações em diferentes plataformas e sistemas operacionais. A portabilidade, segurança, eficiência e recursos avançados de orientação a objetos são apenas alguns dos exemplos de como a linguagem se destaca. Além disso, as funções *Lambda* adicionadas na versão 8 da linguagem oferecem uma série de vantagens importantes para a programação moderna, como a facilidade de escrever código mais legível e modular. Com isso, a linguagem Java continuará sendo uma parte importante e relevante no mundo da programação.

3. RESULTADOS

Foram implementadas funções *Lambda* em Go e Java para calcular a série de *Fibonacci* e medir o tempo de execução usando o tempo de resposta médio (em milissegundos) para 100 solicitações consecutivas e obtemos os seguintes resultados conforme os Quadro 2 e 3.

Quadro 2 – Resultados obtidos com a linguagem Java

ID	Número de elementos da série	Iterativo (ns)	Recursivo (ns)	Memorização (ns)
1	10	36.896	10.938	11.965
2	20	59.270	156.699	32.263
3	30	168.490	3.448.550	23.652
4	40	53.2345	328.152.766	21.307
5	50	747.170	4.523.170.340.963	565.293

Fonte: Elaborado pelos Autores

Quadro 3 – Resultados obtidos com a linguagem GO

ID	Número de elementos da série	Iterativo (ns)	Recursivo (ns)	Memorização (ns)
1	10	286	686	1.524
2	20	300	44.037	2.758
3	30	383	44.764.965	4.629
4	40	698	547.385.755	5.647
5	50	1.180	66.079.079.085	6.727

Fonte: Elaborado pelos Autores

3.1 ANÁLISE DOS RESULTADOS

Com base na tabela, pode-se observar que o desempenho da função *Lambda* em Go é significativamente melhor do que em Java. Isso é evidenciado pelo fato de que os tempos de execução da função iterativa e memorização em Go são em média de duas a três ordens de magnitude menores do que em Java. Além disso, o desempenho da função recursiva em Go também é melhor em comparação com Java, embora não tão significativamente quanto as outras duas funções. Esses resultados sugerem que, para funções *Lambda*, o Go pode ser uma escolha mais eficiente em termos de desempenho em relação ao Java.

4. CONSIDERAÇÕES FINAIS

Como resultado deste estudo, destaca-se a importância da escolha da linguagem de programação adequada para o desempenho de funções *Lambda*. Embora o Go tenha demonstrado ser mais eficiente em termos de desempenho em relação ao Java para a série de *Fibonacci*. É importante ressaltar que este estudo tem algumas limitações que podem afetar a decisão desta escolha. Uma possível limitação refere-se à análise de desempenho apenas para a série de *Fibonacci* e pode não representar adequadamente a complexidade dos problemas do mundo real. Portanto, os resultados dos testes podem não refletir totalmente o desempenho das linguagens em situações mais complexas.

Além disso, é preciso cuidado na medição do tempo de execução, pois pode ser afetada por fatores externos, como a latência da rede e a carga no processador.

Apesar dessas limitações, os resultados são relevantes para o campo de estudo de funções *Lambda* e fornecem informações valiosas para desenvolvedores escolherem a linguagem de programação mais apropriada para suas necessidades e objetivos de desempenho. É importante destacar que a escolha da linguagem de programação ideal deve ser feita considerando diversos fatores, incluindo desempenho, facilidade de desenvolvimento, manutenção do código e disponibilidade de bibliotecas e recursos.

Portanto, recomenda-se que os desenvolvedores avaliem cuidadosamente as suas necessidades e objetivos de desempenho antes de escolher uma linguagem de programação para funções *Lambda*, levando em conta não apenas o desempenho, mas também outros fatores, como a facilidade de desenvolvimento e manutenção do código.

REFERÊNCIAS

AWS. **Estudo de caso da Netflix e do AWS Lambda**. 2014. Neil Hunt - Chief Product Officer Netflix. Disponível em: https://aws.amazon.com/pt/solutions/case-studies/netflix-and-aws-lambda/. Acesso em: 28 maio 2023.

AWS. **Tutorial: Uso do Lambda com API Gateway**. 2023. Disponível em: https://docs.aws.amazon.com/pt_br/lambda/latest/dg/services-apigateway-tutorial.html. Acesso em: 28 maio 2023.

BARENDREGT, **Henk. The impact of the lambda calculus in logic and computer science.** Bulletin of Symbolic Logic, v. 3, n. 2, p. 181-215, 1997.

BIRD, Richard. Introduction to functional programming using Haskell. Pearson Educación, 1998.

DEVMEDIA (org.). **JavaServer Faces: guia de referência jsf - javaserver faces**. Guia de Referência JSF - JavaServer Faces. 2023. Disponível em: https://www.devmedia.com.br/guia/jsf-javaserver-faces/38322. Acesso em: 05 jun. 2023.

DIETZ, Linus et al. **Java By Comparison: Become a Java Craftsman in 70 Examples**. Java By Comparison, p. 1-206, 2018.

DONOVAN, Alan AA; KERNIGHAN, Brian W. **The Go programming language**. Addison-Wesley Professional, 2015.

EYUPOGLU, Can. **Big Data Processing: Concepts, Architectures, Technologies, and Techniques.** In: Applications and Approaches to Object-Oriented Software Design: Emerging Research and Opportunities. IGI Global, 2020. p. 111-132.

FARRELL, Joyce. **Java programming**. Cengage Learning, 2022.

FOUNDATION, **The Apache Software** (org.). Apache Struts. 2023. Disponível em: https://struts.apache.org/. Acesso em: 05 jun. 2023.

GEARY, Cay S. et al. **Java Performance: The Definitive Guide**. Sebastopol: O'Reilly, 2016.

GO, Inc. (org.). **A Tour of Go** - 2023. Disponível em: https://go.dev/tour/concurrency/1. Acesso em: 05 jun. 2023.

ISO/IEC-17025. **International Organisation for Standardisation**. General requirements for the competence of testing and calibration laboratories. 2005.

JANEV, Valentina. **Semantic intelligence in big data applications**. Smart Connected World: Technologies and Applications Shaping the Future, p. 71-89, 2021.

JetBrains. **The State of Developer Ecosystem in 2021 Infographic**. JetBrains: Developer Tools for Professionals and Teams. 2021. Disponível em: https://www.jetbrains.com/lp/devecosystem-2021/#Main. Acesso maio 2023.

KLEMS, Markus. **AWS Lambda Quick Start Guide: Learn how to build and deploy serverless applications on AWS**. Packt Publishing Ltd, 2018.

KORAKITIS, Konstantinos; MUIR, Richard; JONES, Simon; CONDON, Michael. **Developer Economics: State of the Developer Nation** 22nd Edition. 2022. Disponível em: https://developer-economics.cdn.prismic.io/developer-economics/f99dc570-f8f4-41f0-bc3b-808b2dcdb6cb_Slashdata++22nd+edition+of+The+State+of+the+Developer+Nation+%28Q1+2022%29.pdf. Acesso em: 28 maio 2023.

MOORE, Susan. **Gartner Says More Than Half of Enterprise IT Spending in Key Market Segments Will Shift to the Cloud by 2025**. 2022. Disponível em: https://www.gartner.com/en/newsroom/press-releases/2022-02-09-gartner-says-more-than-half-of-enterprise-it-spending. Acesso em: 28 maio 2023.

MUGHAL, Khalid A.; RASMUSSEN, Rolf W. **A Programmer's Guide to Java SE 8 Oracle Certified Associate (OCA)**. Addison-Wesley Professional, 2016.

O'GRADY, Stephen. **The RedMonk Programming Language Rankings: January 2022**.

PATTERSON, Scott. **Learn AWS Serverless Computing: A Beginner's Guide to Using AWS Lambda**, Amazon API Gateway, and Services from Amazon Web Services. Packt Publishing Ltd, 2019.

REDHATE (org.). **Hibernate: everything data**. 2023. Disponível em: https://hibernate.org. Acesso em: 05 jun. 2023.

TINYGO, Inc. (org.). **TINYGO Documentation** - 2023. Disponível em: https://tinygo.org/docs/. Acesso em: 05 jun. 2023.

VMware, Inc. (org.). **SPRING Framework: 6.0.9**. 2023. Disponível em: https://spring.io/projects/spring-framework. Acesso em: 05 jun. 2023.