UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

THIAGO LUIZ RODRIGUES

SIRA: ARQUITETURA PARA RECUPERAÇÃO DE INFORMAÇÃO SEM SERVIDOR

CURITIBA

2023

THIAGO LUIZ RODRIGUES

SIRA: ARQUITETURA PARA RECUPERAÇÃO DE INFORMAÇÃO SEM SERVIDOR

SIRA: SERVERLESS INFORMATION RETRIEVAL ARCHITECTURE

Dissertação de mestrado apresentada como requisito para obtenção do título de Mestre em Computação Aplicada do Programa De Pós-Graduação Em Computação Aplicada - PPGCA da Universidade Tecnológica Federal do Paraná.

Orientador: Profa. Dra. Ana Cristina Barreiras Kochem Vendramin

Coorientador: Prof. Dr. Luiz Nacamura Junior

CURITIBA

2023

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Curitiba

THIAGO LUIZ RODRIGUES

SIRA: ARQUITETURA PARA RECUPERAÇÃO DE INFORMAÇÃO SEM SERVIDOR

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Computação Aplicada da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Engenharia De Sistemas Computacionais.

Data de aprovação: 24 de Novembro de 2023

Dra. Ana Cristina Barreiras Kochem Vendramin, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Luis Carlos Erpen De Bona, Doutorado - Universidade Federal do Paraná (Ufpr)

Dr. Mauro Sergio Pereira Fonseca, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Thiago Henrique Silva, Doutorado - Universidade Tecnológica Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 24/11/2023.

RESUMO

A crescente necessidade de sistemas de recuperação de informação eficientes e escaláveis impulsionou a busca por novas abordagens e arquiteturas. Este trabalho propõe a arquitetura SIRA (*Serverless Information Retrieval Architecture*) para a recuperação de informação sobre a plataforma de computação sem servidor, com foco nas etapas de indexação, busca e classificação. A arquitetura é dividida em duas etapas: a indexação de documentos, que inclui o processamento, extração de métricas e criação de índices invertidos; e a recuperação de documentos, a qual envolve a busca de documentos relevantes, classificação dos documentos através de métricas e a apresentação dos resultados ao usuário. O desempenho da arquitetura SIRA é comparado com o ELS (*Elastic Search*) em duas coleções de teste, *Cranfield* e TREC-COVID. Além de ter a vantagem de consumir recursos apenas sob demanda, a arquitetura SIRA demonstrou um desempenho similar ao ELS, indicando sua eficácia na recuperação de documentos relevantes e seu potencial como uma solução no campo da recuperação da informação.

Palavras-chave: sistemas distribuídos; computação sem servidor; recuperação da informação; indexação de documentos; classificação de documentos.

ABSTRACT

The growing need for efficient and scalable information retrieval systems has driven the search for new approaches and architectures. This work proposes the Serverless Information Retrieval Architecture (SIRA) for information retrieval on the serverless computing platform, focusing on the stages of indexing, searching, and classification. The architecture is divided into two steps: document indexing, which includes processing, metric extraction, and the creation of inverted indexes; and document retrieval, which involves searching for relevant documents, classifying the documents through metrics, and presenting the results to the user. The performance of the SIRA architecture is compared to Elastic Search (ELS) on two test collections, Cranfield and TREC-COVID. In addition to having the advantage of consuming resources only on demand, the SIRA architecture demonstrated similar performance to ELS, indicating its effectiveness in retrieving relevant documents and its potential as a solution in the field of information retrieval.

Keywords: distributed systems; serverless computing; information retrieval; document indexing; document ranking.

LISTA DE ALGORITMOS

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Há 3000 AC, os sumérios já praticavam o ato de arquivar e organizar informações contidas em tábuas de argilas. Eles desenvolveram técnicas para classificar e identificar cada tábua e seu conteúdo com facilidade (SANDERSON; CROFT, 2012). Em 1950 iniciaram as propostas de utilização de computadores para a recuperação da informação (HARMAN *et al.*, 2019).

A recuperação da informação é uma área de pesquisa da ciência da computação que lida com armazenamento, organização e recuperação de documentos de textos não estruturados escritos em linguagem natural (SINGHAL *et al.*, 2001).

Recuperação da informação são técnicas de recuperação de documentos dentro de grandes coleções. As técnicas são classificadas em dois grupos (BUTTCHER; CLARKE; CORMACK, 2016): (1) o primeiro grupo são as técnicas de indexação, como: pré-processamento, extração de métricas, criação de índice invertido; (2) o segundo grupo envolve os algoritmos de busca e classificação que tem como objetivo escolher os documentos mais relevantes para a busca do usuário.

Sistemas de recuperação da informação são softwares que empregam as técnicas de recuperação da informação. O sistema de recuperação da informação mais famoso entre o usuários da *web* é o *Google*.

Os sistemas de recuperação da informação também utilizam técnicas de sistemas distribuídos, como: distribuição do processamento entre serviços, comunicação assíncrona através de eventos, escalabilidade horizontal e uso de cache. O objetivo de utilizar técnicas de sistemas distribuídos na recuperação da informação é para que os sistemas possam retornar documentos de forma mais rápida aos usuários (BATES, 2011).

Porém, configurar e gerir uma infraestrutura para suportar sistemas distribuídos é algo complexo, pois evolve gestão de servidores, balanceador de carga, serviços de resolução de nomes e rede.

Alguns estudos, como os de Cacheda, Plachouras e Ounis (2005) e Bashir *et al.* (2013), têm empregado a computação provisionada como arquitetura para que os sistemas de recuperação da informação tenham um melhor desempenho.

A computação provisionada é um modelo de computação em nuvem onde o usuário precisa alocar recursos de infraestrutura de Tecnologia da Informação (TI) para executar sua aplicação. Isso envolve a escolha, configuração e gestão dos servidores físicos e virtuais (SEDEFOĞLU; SÖZER, 2021). Um exemplo de uma arquitetura voltada para a recuperação da informação que funciona sobre a plataforma de computação provisionada é o *Elastic Serch* (ELS).

Conforme aumenta a coleção de documentos de um sistema de recuperação da informação, milhares de computadores (recursos) são necessários para atender a carga de trabalho e entregar documentos rapidamente aos usuários (FREIRE *et al.*, 2013). Provisionar esses novos recursos na camada de nuvem não é uma tarefa trivial.

Faz-se, então, necessária uma arquitetura que seja capaz de prover novos recursos quando ocorrerem picos de acessos inesperados ou o aumento na coleção de documentos. Diante desse desafio, a computação sem servidor parece ser uma estratégia viável em relação ao uso da computação provisionada na recuperação da informação, pois não é preciso alocar uma infraestrutura de TI para executar as aplicações (CRANE; LIN, 2017).

O termo computação sem servidor se refere ao fato de desenvolvedores de *software* poderem se concentrar em regras de negócios das aplicações sem se preocupar com questões relacionadas à infraestrutura, configurações e escalabilidade (ANDI, 2021). A computação sem servidor usa a arquitetura orientada a eventos, pois as aplicações são executadas mediante um evento assíncrono ou síncrono (LI *et al.*, 2022).

Em 2017 iniciaram os estudos que adaptavam os sistemas de recuperação da informação para serem utilizados sobre a plataforma de computação sem servidor. O primeiro estudo sobre o uso da computação sem servidor para suportar sistemas de recuperação da informação foi realizado por Crane e Lin (2017). Outro estudo foi feito por Anand *et al.* (2021) no qual os autores adaptaram uma biblioteca para construção de sistemas de recuperação da informação feita para computação provisionada, para ser utilizada sobre a computação sem servidor. Porém, nenhum trabalho até o momento abordou o tema de forma completa, envolvendo as etapas de indexação, busca e classificação. Surge, então, uma questão de pesquisa: é possível que um sistema de recuperação da informação funcione sobre a computação sem servidor e seja capaz de executar as etapas de indexação, busca e classificação?

1.1 OBJETIVO GERAL E OBJETIVOS ESPECÍFICOS

O objetivo desse trabalho é propor uma arquitetura para recuperação da informação, chamada SIRA (*Serverless Information Retrieval Architecture*), que funcione sobre a plataforma de computação sem servidor e que aborde as etapas de indexação, busca e classificação.

O detalhamento do objetivo geral é apresentado na forma dos seguintes objetivos específicos:

- Construir uma biblioteca para extração de métricas de documentos;
- Implementar um algoritmo probabilístico para classificação de documentos;
- Analisar o desempenho das arquiteturas SIRA e ELS em relação à recuperação de documentos;
- Analisar o desempenho das arquiteturas SIRA e ELS em relação ao consumo de recursos durante a recuperação de informações.

1.2 ESTRUTURA DO DOCUMENTO

Este documento está organizado em cinco capítulos. O Capítulo 1 apresenta a introdução, objetivos e contribuições deste trabalho. O Capítulo 2 apresenta a fundamentação teórica que descreve com mais detalhes a computação sem servidor, recuperação da informação e os trabalhos relacionados. O Capítulo 4 apresenta a arquitetura proposta. No Capítulo 5 são apresentados os resultados obtidos. Finalizando, o Capítulo 6 traz as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

O objetivo deste capítulo é detalhar os conceitos que permitam compreender a arquitetura proposta neste trabalho para um sistema de recuperação da informação que funcione sobre a plataforma de computação sem servidor. Esses conceitos envolvem a Computação em Nuvem (Seção 2.1), a Arquitetura Orientada a Eventos (Seção 2.2), a Computação sem Servidor (Seção 2.3) e a Recuperação da Informação (Seção 2.4). Ao final do capítulo, são apresentados os trabalhos relacionados (Seção 3).

2.1 Computação em Nuvem

O conceito de computação em nuvem não é novo. A ideia foi sugerida pela primeira vez por John Mc Carthy na década de 1960, quando ele propôs o conceito de "*Utility Computing*" ou computação utilitária. A computação em nuvem de certa forma é um grande pacote de utilitários, em vez de um produto ou tecnologia específica. A computação em nuvem é uma evolução da computação em grade (MENKEN, 2008). A computação em grade é um sistema de computação paralela e distribuída geograficamente que permite o compartilhamento de recursos (BUYYA *et al.*, 2009).

A motivação por trás da computação em grade era resolver o problema de escalonar aplicações de grande porte, pois, um único servidor não seria suficiente para suportar uma grande aplicação (CASANOVA, 2002). Em vários momentos a computação em nuvem pode ser confundida com computação em grade. As distinções não são claras porque ambas compartilham visões semelhantes: reduzir custos de computação e aumentar a flexibilidade e confiabilidade usando *hardware* operado por terceiros (VAQUERO *et al.*, 2008).

Na computação em grade, a aplicação é subdividida em unidades menores que serão executadas em vários servidores (CASANOVA, 2002). A computação em nuvem faz exatamente o oposto. Ela permite que vários aplicativos executem ao mesmo tempo no mesmo servidor, tudo isso graças à virtualização. As nuvens são mais populares do que as grades devido aos seus serviços amigáveis, virtualizados e escalonáveis automaticamente. Ao contrário da computação em grade, a nuvem permite o provisionamento de recursos em tempo real sempre que a aplicação tiver necessidade de mais recursos (GOYAL; DADIZADEH, 2009).

O NIST (*National Institute of Standards and Technology*) descreve cinco características essenciais da computação em nuvem (YANG; HUANG, 2013):

- Elasticidade: é definida como a capacidade de aumentar ou diminuir os recursos conforme necessário;
- Serviço gerenciado: os serviços em nuvem são controlados e monitorados pelo provedor de nuvem. Isso é crucial para faturamento, controle de acesso, otimização de recursos, planejamento de capacidade e outras tarefas;

- Autoatendimento: significa que um consumidor pode usar os serviços em nuvem conforme necessário, sem nenhuma interação humana com o provedor de nuvem;

- Acesso de forma onipresente: significa que os recursos do provedor de nuvem estão disponíveis na rede e podem ser acessados;

- *Pooling* de recursos: permite que um provedor de nuvem atenda seus consumidores por meio de um modelo multi-locatário. O cliente geralmente não tem controle ou conhecimento sobre a localização exata dos recursos fornecidos, mas, pode ser capaz de especificar a localização em um nível mais alto de abstração (por exemplo, país, estado ou centro de dados).

Os serviços em nuvem podem reduzir o custo e a complexidade de gerenciar servidores e redes. Usuários de serviço em nuvem não precisam investir em infraestrutura, *hardware* ou comprar licenças de *software*. Os benefícios são os baixos custos dos serviços, retorno rápido sobre o investimento e implantação rápida (GOYAL; DADIZADEH, 2009). Além disso, os provedores de nuvem possuem departamentos de pesquisas com o objetivo de trazer inovações que podem ser usadas pelos clientes (BUYYA *et al.*, 2009).

Outros benefícios da computação em nuvem para os usuários incluem escalabilidade, confiabilidade e eficiência (GOYAL; DADIZADEH, 2009). Escalabilidade significa que a computação em nuvem oferece processamento e capacidade de armazenamento ilimitados. A nuvem é confiável na medida em que permite o acesso a aplicativos e documentos em qualquer lugar do mundo através da Internet (GOYAL; DADIZADEH, 2009).

O objetivo final da computação em nuvem é fornecer aplicativos, plataformas e infraestrutura como serviços.

O *NIST* define três modelos de prestação serviços na nuvem, amplamente conhecidos como “*as a services*” (YANG; HUANG, 2013):

- *Software* como serviço (SaaS). Nesse modelo, o cliente apenas utiliza um *software* em específico através de um modelo de assinatura. Assim, o cliente permanece livre da atualização e manutenção do *software*. O SaaS possui limitação, pois uma empresa pode não encontrar aplicativos específicos. Além disso, se um cliente desejar trocar de fornecedor, em muitos casos não vai conseguir migrar os dados de um *software* para o outro, gerando um bloqueio de fornecedor (*vendor lock-in*);

- Plataforma como serviço (PaaS). Nesse modelo, os clientes desenvolvem, executam e gerenciam suas aplicações sem a necessidade de construir e manter uma infraestrutura na nuvem. O modelo PaaS entrega uma plataforma para executar o *software* e ferramentas de desenvolvimento. Nesse modelo, as empresas reduzem os custos com licenças de *software* e infraestrutura. O *Kubernates* é um serviço disponibilizado de forma PaaS pelos provedores de nuvem (GOYAL; DADIZADEH, 2009);

- Infraestrutura como serviço (IaaS). É um modelo de serviço de computação em nuvem que oferece recursos essenciais de computação, armazenamento e rede sob demanda, com pagamento conforme o uso.

2.2 Arquitetura Orientada a Eventos

Acoplamento é algo altamente discutido na engenharia de software, pois, o alto acoplamento é a causa de muitos projetos falharem quando submetidos ao ambiente de produção. Na engenharia de software, o acoplamento é o grau de interdependência entre os módulos de software. Então, dois módulos fortemente acoplados são fortemente dependentes um do outro (EDER; KAPPEL; SCHREFL, 1994). Um exemplo de acoplamento forte são os *softwares* que estão distribuídos, mas que utilizam uma abordagem de comunicação síncrona. Essa abordagem é um problema, pois, se o serviço (A) necessita acessar um recurso do serviço (B) e o serviço (B) falha, isso faz com o que o serviço (A) também falhe.

A Arquitetura Orientada a Eventos (EDA - *Event Driven Architecture*) vem sendo utilizada em casos onde o alto acoplamento pode trazer prejuízos para um negócio (DAHAN, 2009). A EDA é caracterizada pela existência de serviços independentes que geram eventos com a finalidade de atingir um objetivo coordenado, através de uma comunicação assíncrona (ROCHA; ROCHA, 2022).

Eventos são acontecimentos que representam uma mudança de estado (ROCHA; ROCHA, 2022). Um evento é um tipo de mensagem produzida por um serviço para transmitir informações específicas. No corpo de uma mensagem são inseridas informações sobre o evento realizado, como exemplos: “cliente cadastrado”, “venda efetuada”, “status do pedido alterado”. Esse tipo de mensagem tem como característica não ser destinado a um serviço específico.

Além de mensagens do tipo eventos, também existe um modelo de mensagem do tipo comando. Comando representa uma intenção. Ele tem como alvo um serviço em específico. Exemplo: serviço (A) envia um comando “altere o status do pedido para concluído” para o serviço (B). O serviço que gera o comando deve conhecer o serviço que vai executar o comando (ALEXANDER, 2022).

Em uma comunicação assíncrona orientada a eventos, um serviço, chamado de produtor, publica eventos em um serviço de mensagens (*message broker*). Muitos outros serviços, chamados de assinantes ou consumidores, podem se inscrever nesse serviço de mensagens para serem notificados e agirem sobre um evento (MICHELSON, 2006). Produtores e consumidores se comunicam com o serviço de mensagens através de protocolos de comunicação que podem ser abertos ou proprietários. Alguns exemplos de protocolos de comunicação são: (MQTT - *Message Queue Telemetry Transport*), (STOMP - *Simple Text Oriented Message Protocol*), (CoAP - *Constrained Application Protocol*) e (AMQP - *Advanced Message Queuing Protocol*) (THANGAVEL *et al.*, 2014) (O’HARA, 2007).

O serviço que gera o evento não tem conhecimento sobre os serviços consumidores, o que gera um baixo acoplamento entre serviços. Portanto, se o produtor de um determinado evento falhar, não vai prejudicar os serviços responsáveis por consumir os eventos (ALEXANDER, 2022). O serviço de mensagens será o responsável pela entrega da mensagem para os serviços que tem o interesse no evento.

Ao receber mensagens de notificação de eventos, os assinantes avaliam a mensagem e, opcionalmente, podem executar uma ação. Por exemplo, na ocorrência de um evento de venda efetuada com sucesso, os consumidores desse evento podem executar a ação de retirar o item do estoque e gerar uma separação para entrega do produto (DAHAN, 2009).

Existem duas formas de publicar mensagens em um serviço de mensagens (ALEXANDER, 2022): através de filas ou do modelo *publish/subscribe* (publicar/assinar).

No modelo *publish/subscribe*, os produtores enviam mensagens do tipo evento para o *broker* e essas mensagens ficam organizadas em tópicos. Os tópicos são um meio de classificar as informações e, na prática, são simplesmente texto.

No modelo de filas, um ou mais produtores podem enviar mensagens para uma fila no serviço de mensagens. Porém, o serviço de mensagens vai entregar a mensagem para somente um consumidor (ALEXANDER, 2022).

2.3 Computação sem Servidor

O modelo de computação sem servidor (*serverless*) foi criado pela *Amazon* em 2014 (HASSAN; BARAKAT; SARHAN, 2021)(BALDINI *et al.*, 2017). Em 2016, empresas como *Google* e *Microsoft* adotaram o modelo. A computação sem servidor abstrai a complexidade de gestão, administração e configuração dos modelos IaaS e PaaS (BALDINI *et al.*, 2017).

O modelo de computação sem servidor (*serverless*) promete implantação rápida e de baixo custo com operação sem gerenciamento (CASTRO *et al.*, 2017). A computação sem servidor teve um crescimento rápido. Atualmente, existem dezenas de plataformas de computação sem servidor e todos os provedores globais de nuvem oferecem esse serviço.

A computação sem servidor é um modelo de computação em nuvem que permite aos usuários executar *softwares* orientadas a eventos, sem ter que lidar com o provisionamento, configuração e gerenciamento da infraestrutura de servidores, sendo cobrado somente pelo tempo que o *software* estiver em execução (BALDINI *et al.*, 2017)(ROBERTS; CHAPIN, 2017).

A computação sem servidor (*serverless*) é uma evolução do modelo PaaS (plataforma como serviço). No modelo de computação sem servidor não é necessário dizer quanto de memória ou processamento deve existir no servidor para suportar uma aplicação. Não sendo necessário qualquer tipo de configuração de provisionamento e com um modelo de escalonamento totalmente transparente. (EYK *et al.*, 2018).

Aplicativos desenvolvidos para a computação sem servidor na nuvem também são conhecidos como funções, por sua característica *stateless* (sem estado), pois as aplicações não

mantêm nenhum estado. A memória é alocada para execução e após o término da execução toda memória alocada é liberada, como a execução de uma função em uma linguagem de programação qualquer (CASTRO *et al.*, 2017).

As funções são iniciadas através de eventos. Então, para que uma função seja executada é necessário que algum evento ocorra. Esses eventos podem ser uma mensagem publicada em um tópico, um arquivo adicionado a um serviço de repositório, uma solicitação via HTTP (*Hypertext Transfer Protocol*) (CASTRO *et al.*, 2017).

Cada fornecedor de nuvem possui seu modelo de arquitetura para computação sem servidor. Os principais fornecedores dos serviços de computação sem servidor na nuvem são:

- *Amazon AWS*: serviço de computação sem servidor chamado *Lambdas Functions*;
- *Microsoft Azure*: serviço chamado *Azure Functions*;
- *Google Cloud Platform*: serviço chamado *Cloud Functions*.

No presente trabalho é utilizada a arquitetura *AWS Lambdas Functions* da *Amazon*, a qual está detalhada na próxima seção.

2.3.1 Arquitetura *Lambdas Functions*

Quando a arquitetura *AWS Lambda* foi construída, inicialmente foi escolhido o modelo virtualizado de servidor junto a contêineres *Linux*. Cada conta de cliente possuía uma máquina virtual (do inglês *Virtual Machine* (VM)) EC2 (*Elastic Compute Cloud*) e o contexto de execução das funções era isolado por contêiner *Linux* (AGACHE *et al.*, 2020). Porém, existe um problema de segurança nessa abordagem utilizando contêineres, pois se confia todo o contexto de execução das funções a um único *Kernel* de sistema operacional, no qual executa-se código de funções não seguras. Segundo (AGACHE *et al.*, 2020), as chamadas ao sistema (*systemcalls*) poderiam ser limitadas o que melhoraria a segurança ao executar um código não seguro, porém isso restringiria os recursos disponíveis às funções.

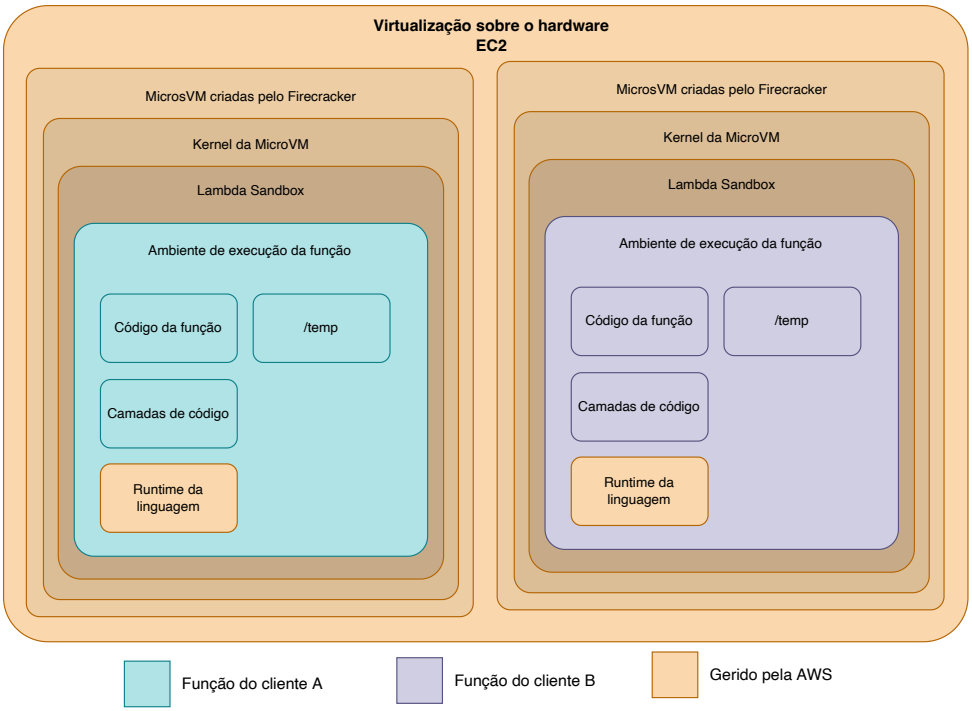
A virtualização é o caminho natural para a solução dos problemas de compartilhamento do mesmo *kernel* de sistema operacional com diversas outras funções, pois cada função poderia ter o seu próprio *kernel* isolado para execução de código não seguro. Pesquisadores da AWS procuraram uma solução de virtualização que pudesse gerar um isolamento forte na execução das funções, uma rápida inicialização e que executasse milhares de funções em um mesmo servidor sem desperdícios. Diante dessa demanda, nasceu o projeto *Firecracker* (AGACHE *et al.*, 2020).

O *Firecracker* é um monitor de máquina virtual (do inglês *Virtual Machine Monitor* (VMM)) que usa a infraestrutura de virtualização do KVM (*Kernel Virtualization Monitor*), isto é, uma máquina virtual baseada em *kernel*) do Linux (AGACHE *et al.*, 2020). Um VMM é um

software que permite a criação e gerenciamento de VMs, e gerencia a operação de um ambiente virtualizado em cima de uma máquina física (AZMANDIAN *et al.*, 2011). O VMM gerencia a operação de *back-end* das VMs, alocando memória, armazenamento e outros recursos de entradas e saídas necessários (AZMANDIAN *et al.*, 2011).

Cada função executa dentro de uma micro VM criada e gerenciada pelo *Firecracker*. As micro VM são compostas por *sandbox*, que são ambientes de execução de funções, como pode-se observar na Figura 1. As *micros VM* são pequenas VMs que possui um micro *kernel*, contendo somente o necessário para executar funções, sendo removido acesso ao USB (*Universal Serial Bus*), vídeo, entrada serial, etc. As micros VM são iniciadas em um tempo médio de 125ms. O ambiente de execução das funções é criado dentro de *workers*. *Workers* são instâncias EC2 não visualizadas pelos usuários dos serviços *Lambdas Functions*. Os *workers* suportam a execução de milhares de micro VMs.

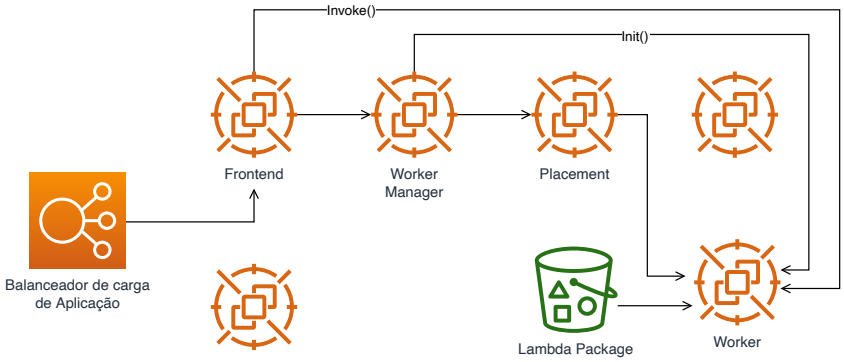
Figura 1 – Composição de um *Worker*.



Fonte: Adaptado de Agache *et al.* (2020).

As funções podem ser invocadas através de solicitações síncronas ou assíncronas. Como pode ser visto na Figura 2, quando acontece uma solicitação síncrona de execução de uma função, a solicitação passa pelo balanceador de carga (*load balancer*) que tem a função de equilibrar as chamadas entre diversos servidores (SCHAATSBERGEN, 2021). O balanceador de carga (*load balancer*) encaminha a solicitação para uma camada de *front-end* que tem a função de verificar se quem solicitou a execução da função tem permissão para executar. O *front-end* também verifica se a função atingiu a quantidade máxima de execuções simultâneas. Após as validações, o *front-end* encaminha a solicitação para o *worker manager* que tem a função de verificar se existe um ambiente de execução de funções (*sandbox* ativos).

Figura 2 – Execução síncrona de funções.

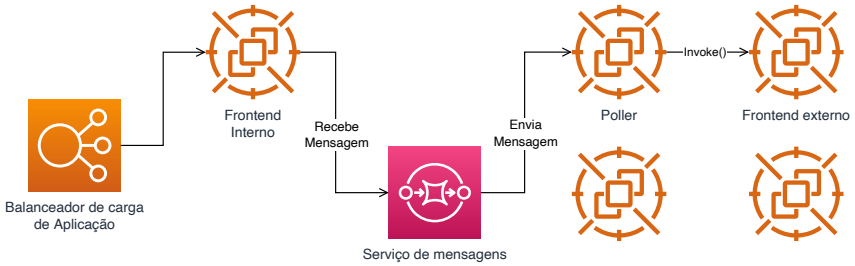


Fonte: Adaptado de Schaatsbergen (2021).

Se for a primeira execução da função, não vão existir *sandboxs* disponíveis para execução. O *worker manager* se comunica com o *placement service* que é responsável por iniciar uma *sandbox* dentro de um *worker*. Durante o processo de criação de uma *sandbox*, é feito o *download* do código de execução das funções. Após iniciada uma *sandbox* pode-se, então, executar uma função.

Já nas execuções assíncronas das funções, como pode-se ver na Figura 3, o balanceador de carga encaminha a solicitação para o *front-end* externo, que a publica em um tópico de um serviço de mensagens. Existe um componente chamado *poller* que recebe os eventos e repassa para o *front-end* interno. A partir desse momento, o fluxo de comunicação seguirá conforme a chamada de invocação síncrona descrita na Figura 2.

Figura 3 – Execução assíncrona de funções.



Fonte: Adaptado de Schaatsbergen (2021).

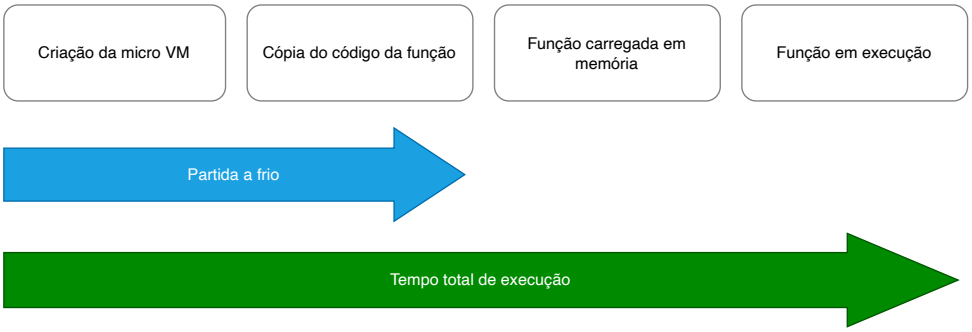
2.3.2 Partida a Frio

Um recurso exclusivo da computação sem servidor é a capacidade de escalar as aplicações sem necessidade de alocar recursos previamente pelo criador da função (AL-MASRI *et al.*, 2018). Essa facilidade gerou alguns desafios como a partida a frio (*cold start*). A partida a frio é o tempo necessário para criação de um ambiente de execução de funções (VAHIDINIA; FARAHANI; ALIEE, 2020).

Quando uma função for executada através de algum evento e seu ambiente de execução não estiver ativo, é preciso iniciar um ambiente de execução para a função. Durante essa

etapa, o serviço faz a criação de uma micro VM, logo após efetua o *download* do código da função, o qual é armazenado em um repositório interno. Em seguida, a função é carregada em memória. Depois de concluído, é possível executar a função, como pode-se observar na Figura 4 (VAHIDINIA; FARAHANI; ALIEE, 2020).

Figura 4 – Etapas para execução de uma função.



Fonte: Adaptado de Vahidinia, Farahani e Aliee (2020).

Existem duas abordagens gerais para lidar com atrasos na inicialização das funções (SUO *et al.*, 2021): a primeira abordagem tenta minimizar o atraso da partida a frio (otimizando ambientes) e a segunda abordagem minimiza a frequência de ocorrência da partida a frio (*Pinging*).

O objetivo da primeira abordagem é minimizar a duração da partida a frio através de duas soluções: reduzir o atraso na criação do ambiente de execução de funções e reduzir o atraso no carregamento das bibliotecas de funções.

A segunda abordagem tem a função de evitar a recriação de ambientes de execuções de funções com frequência. Após a execução de uma função, plataformas como *Google Cloud Functions* e *Lambdas Functions* mantêm o ambiente de execução ativo para responder a quaisquer solicitações subsequentes. Por exemplo, a plataforma do *Google* mantém 85% dos contêineres aquecidos por até 5 horas e este tempo para o *Lambda Functions* é entre 15 minutos a 5 horas (HASSAN; BARAKAT; SARHAN, 2021).

2.4 Recuperação da Informação

Após os computadores serem inventados, as pessoas perceberam que eles poderiam ser usados para armazenar e recuperar mecanicamente grandes quantidades de informações (SANDERSON; CROFT, 2012). Vannevar Bush publicou em 1945, um artigo com o título "*As We May Think*" que deu origem à ideia de acesso automático a grandes quantidades de conhecimento armazenado (MCGUIRK, 2007).

Na década de 1950, vários trabalhos surgiram que colaboraram com a ideia básica de pesquisar textos com um computador. O impacto dos computadores em recuperação da informação foi percebido quando Hollywood chamou a atenção do público, mostrando a inovação

na comédia *Desk Set*, lançada em 1957. A Comédia mostrava um grupo de bibliotecários que estava prestes a ser substituído por um computador (SANDERSON; CROFT, 2012).

Em 1957, HP Luhn descreve um dos métodos mais influentes, em que ele propôs usar palavras como unidades de indexação para documentos e medir a sobreposição de palavras como critério de recuperação (SANDERSON; CROFT, 2012). Em Jr (1953) foi discutido um projeto para usar o computador Univac para pesquisar 1.000.000 de registros indexados por assunto, estimando-se que levaria 15 horas para pesquisar tantos registros.

Nas décadas de 1970 e 1980 várias técnicas de recuperação de documentos foram desenvolvidas e avanços foram feitos em todas as dimensões do processo de recuperação da informação. Porém, devido à falta de grandes coleções de documentos, ficou a pergunta se essas técnicas seriam capazes de trabalhar com um volume maior de documentos.

Esse panorama mudou em 1992 com o início da Conferência de Recuperações de Textos (TREC - *Text Retrieval Conference*) (SANDERSON; CROFT, 2012). Com grandes volumes de documentos disponíveis no TREC, várias técnicas antigas foram modificadas e outras foram desenvolvidas para recuperar informação de grandes coleções de documentos. O TREC também ramificou a recuperação da informação em campos relacionados, como recuperação de informações faladas, recuperação de idioma diferente do inglês. Os algoritmos de recuperação da informação desenvolvidos foram usados para construção de sistemas de busca na *web* (SALTON, 1971) (SALTON; MCGILL, 1983) (SANDERSON; CROFT, 2012).

2.4.1 Arquitetura de Sistemas de Recuperação da Informação

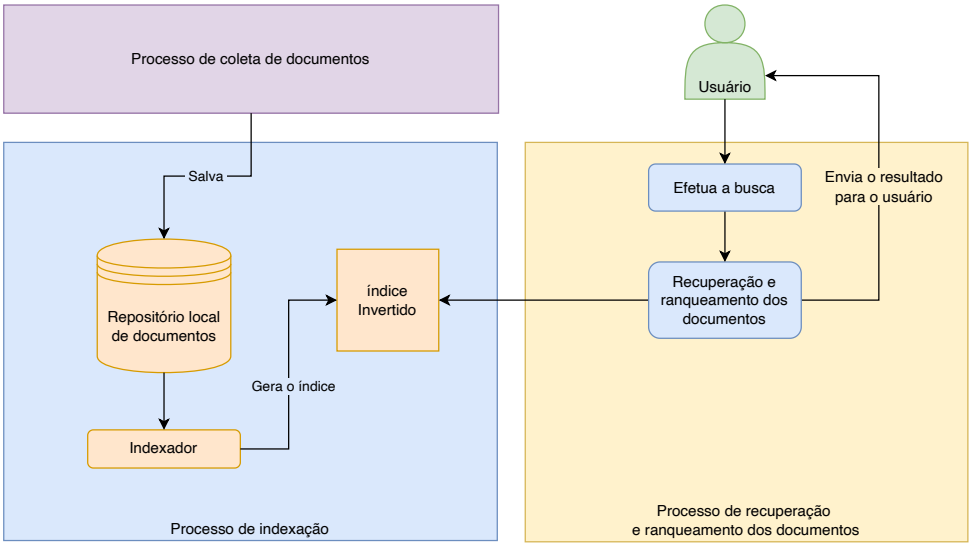
O objetivo de um sistema de recuperação da informação é retornar uma lista de documentos relevantes através de uma busca não estruturada feita em linguagem natural (linguagem humana) pelo usuário. O sistema precisa retornar documentos relevantes, mesmo que os termos que o usuário utiliza na consulta não esteja presentes nesses documentos (SINGHAL *et al.*, 2001).

Já na recuperação de dados, a consulta é estritamente de acordo com a especificação do usuário. Em outras palavras, essas ferramentas pressupõem que os usuários sabem exatamente o que desejam e que suas necessidades de informação são expressas com precisão pela consulta (BAEZA-YATES; RIBEIRO-NETO, 2013). Segundo Singhal *et al.* (2001), uma grande diferença entre sistemas de recuperação de informação e sistemas de recuperação de dados é a natureza da incerteza no processo de busca.

As técnicas de recuperação de informação foram projetadas para resolver os problemas de busca em uma grande coleção de documentos de texto não estruturados escritos em linguagem natural, como artigos ou livros em uma biblioteca e, posteriormente, páginas na web. Sistemas de recuperação da informação precisam de processos para interpretar uma consulta feita em linguagem natural e retornar documentos relevantes para o usuário (BAEZA-YATES; RIBEIRO-NETO, 2013).

A Figura 5 mostra um modelo genérico de um sistema de recuperação da informação. Do lado esquerdo da imagem ficam os processos de coleta de documentos e indexação. Do lado direito fica o processo de recuperação e ranqueamento de documentos baseado em uma busca executada pelo usuário.

Figura 5 – Arquitetura genérica de um sistema de recuperação da informação.



Fonte: Adaptado de Baeza-Yates e Ribeiro-Neto (2013).

O processo de coleta de documentos é a etapa onde se buscam documentos de algum lugar, seja em uma pasta no computador, em uma rede local, na web, etc. Esses documentos geralmente estão desorganizados. Sua principal função é tratar esses documentos e organizá-los dentro de um repositório do sistema de recuperação da informação. Após os documentos estarem organizados, inicia-se a etapa do Indexador. Primeiramente, os documentos que estão no repositório local são pré-processados e nesse processo são removidas palavras irrelevantes que também são chamadas de *stop words*. Após essa etapa é criado um índice global de palavras, onde as palavras não se repetem. Cada palavra do índice aponta para os documentos que possuem aquela palavra, esse processo é chamado de índice invertido (BAEZA-YATES; RIBEIRO-NETO, 2013).

A criação de um índice invertido reduz a dimensionalidade dos dados, fazendo com que as consultas tenham um melhor desempenho.

Do lado direito da Figura 5 fica o processo de recuperação e ranqueamento de documentos. Primeiramente, o usuário efetua uma busca em linguagem natural. O texto da busca é transformado em um vetor de palavras e, após isso, o sistema acessa o índice invertido e compara a palavra do índice com as palavras do vetor. Quando existe correspondência, o sistema obtém os documentos para os quais a palavra aponta. Por fim, cada documento recuperado é avaliado para descobrir o seu nível de similaridade com a busca do usuário (ranqueamento). Normalmente, essa avaliação depende de um algoritmo de classificação que calcula um valor para cada documento. Um documento com um valor maior é considerado mais relevante (MANNING, 2008). Em seguida, os documentos recuperados são retornados em ordem decrescente

de acordo com os resultados do algoritmo de classificação (BAEZA-YATES; RIBEIRO-NETO, 2013) (SALTON, 1971).

2.4.2 Processamento de Texto em Linguagem Natural

Em sistemas de recuperação da informação as buscas são feitas em linguagem natural e os documentos indexados também estão escritos em linguagem natural. Para que seja possível a construção de sistemas moderno de recuperação da informação, os pesquisadores recorreram a uma área da ciência da computação chamada Processamento de Linguagem Natural (). O objetivo da PNL é realizar o processamento de texto escrito em linguagem natural (LIDDY, 2001).

O PLN utiliza o processamento computacional para fazer a compreensão de linguagens humanas. Desde a década de 1980, o campo depende cada vez mais da ciência de dados, estatística, probabilidade e aprendizado de máquina (BUTTCHER; CLARKE; CORMACK, 2016). Com aumento no poder computacional, paralelização e unidades de processamento gráfico (GPU - *Graphics Processing Unit*), foi possível aplicar técnicas de inteligência artificial no processamento de linguagem natural (BUTTCHER; CLARKE; CORMACK, 2016).

A primeira etapa de um PLN é o pré-processamento, o qual tem o objetivo de remover ruídos de um texto, como palavras que sem elas ainda é possível entender o texto. Esse processo é detalhado na Seção (2.4.2.1). Outra etapa importante de um PLN é a extração de métricas do texto, a qual obtém informações que ajudam o sistema de recuperação da informação a entregar para o usuário os documentos mais significantes. Esse processo é descrito na Seção (2.4.2.2).

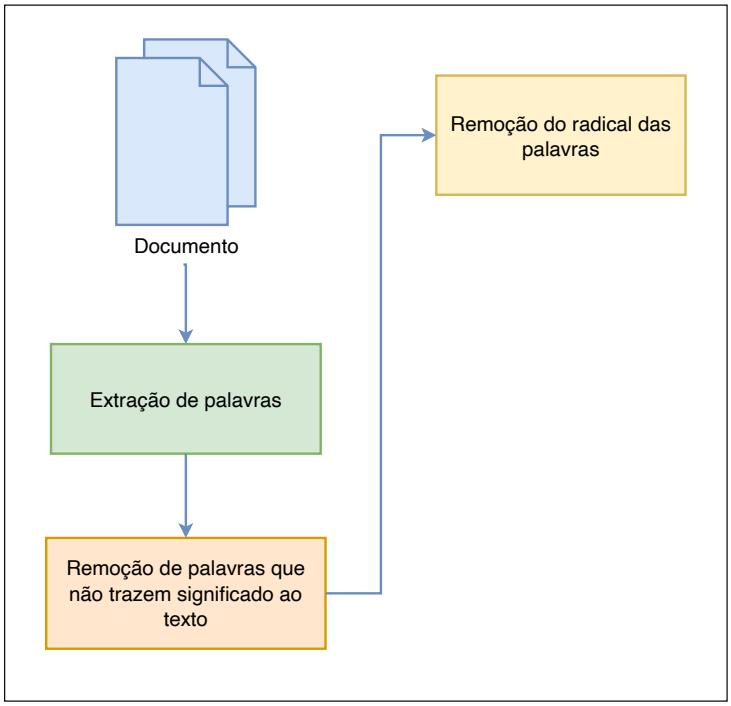
2.4.2.1 Pré-processamento

O pré-processamento é o processo de limpeza e preparação do texto para classificação. Essa etapa ajuda a reduzir os ruídos no texto e melhora o desempenho do processo de busca e classificação de texto (HADDI; LIU; SHI, 2013). A primeira etapa do pré-processamento é a tokenização, um processo de identificação de símbolos (*tokens*) em documentos de textos escritos em linguagem natural (BUTTCHER; CLARKE; CORMACK, 2016).

As etapas do processo de tokenização são mostradas na Figura 6.

Primeiramente, é feito o processo de extração de palavras de um documento, onde é gerado um vetor de palavras "*[Olá, como, vai, você, ?]*" do texto "*Olá como vai você?*". Após a extração de palavras, inicia-se o processo para remoção de palavras que não trazem significado ao texto, tais como artigos, conjunções e pronomes. Esse processo também é conhecido como *stopwords*. Exemplo: "*Maria anda de bicicleta a noite*" se transforma em "*Maria anda bicicleta noite*". Esse processo reduz o tamanho de um texto. Após a remoção de algumas palavras, inicia-se o processo de remoção do radical das palavras, também conhecido como (*stemming*).

Figura 6 – Etapas do pré-processamento.



Fonte: Adaptado de Singh e Saini (2014).

Caso o texto tenha palavras como: "*Andar Andando*" é removido o seu radical "*r, ndo*" e ambas as palavras são transformadas em "Anda". O principal papel do *stemming* é remover vários sufixos para que haja mais redução no número de palavras (SINGH; SAINI, 2014).

Pode-se observar que após a saída do processo mostrada na Figura 7, os documentos são transformados em vetor(*array*) de palavras, e existe uma certa redução de dimensionalidade de cada texto.

Para o desenvolvimento deste trabalho, optou-se por utilizar a técnica de *Tokenização por espaço em branco*, a qual é uma das abordagens mais simples para dividir um texto em *tokens*. Esta técnica funciona bem para idiomas que usam espaços para separar palavras, como é o caso do inglês e do português (THANAKI, 2017). No entanto, existem outras técnicas de tokenização que também podem ser relevantes, dependendo do contexto e do idioma em questão (THANAKI, 2017):

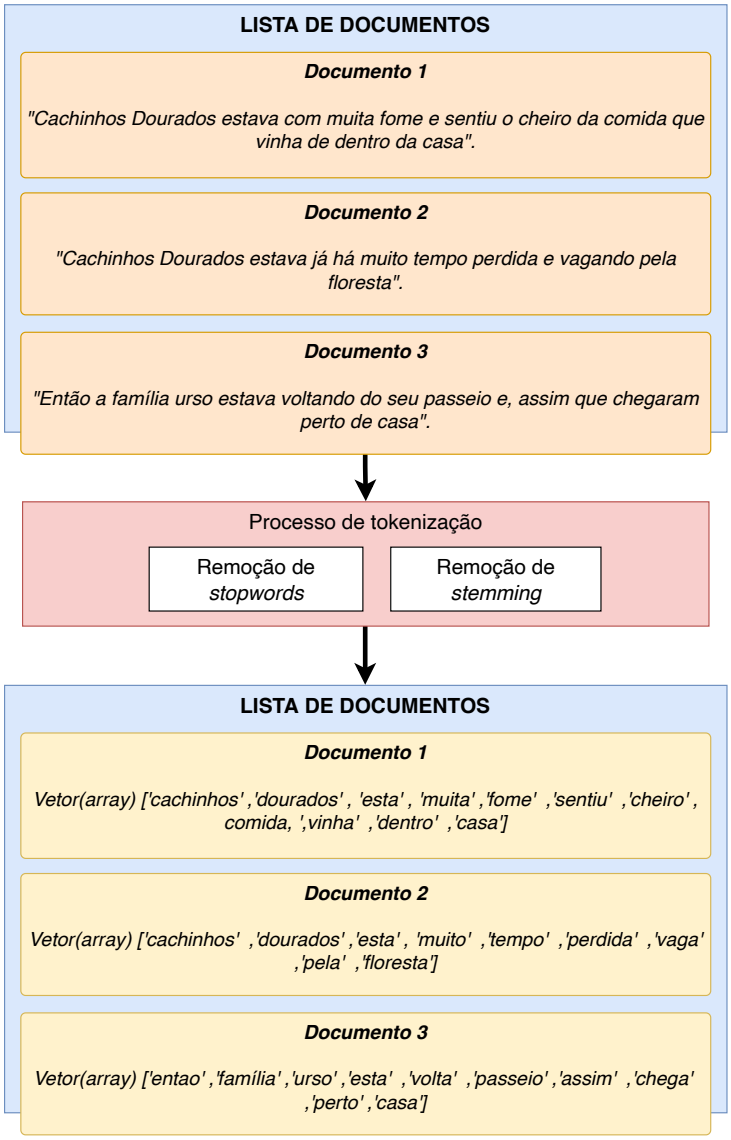
- **Tokenização por Pontuação**: nesta técnica, o texto é dividido em *tokens* com base na pontuação, além dos espaços em branco. Essa abordagem é útil para idiomas que fazem uso consistente de pontuação, como o inglês e o português, e é muitas vezes combinada com a tokenização por espaço em branco para um resultado mais preciso;
- **Tokenização por Expressões Regulares**: esta técnica utiliza expressões regulares para segmentar o texto em *tokens* e oferece uma grande flexibilidade. É especialmente útil para tratar de cenários específicos que outras técnicas mais simples não conse-

guem abordar, não estando necessariamente ligada a idiomas com uso inconsistente de pontuação;

- **Tokenização Baseada em Morfologia**: esta técnica emprega regras morfológicas para a segmentação do texto. É particularmente útil para idiomas com estruturas morfológicas complexas, como o árabe. Em relação ao chinês, que frequentemente utiliza caracteres únicos para representar palavras inteiras, outras técnicas específicas de tokenização seriam mais adequadas.

É importante notar que a eficácia de cada técnica de tokenização pode depender do contexto específico de sua aplicação, variando conforme o idioma, o domínio do texto e os objetivos do processamento (MANNING, 2008).

Figura 7 – Processo de pré-processamento e pós-processamento.



Fonte: Autoria própria.

2.4.2.2 Métricas de um Texto

A extração de métricas (estatística) de um texto é uma das técnicas de PNL e classificação de texto. Em recuperação da informação, esses dados podem ser utilizados para definir quão relevante é um documento para uma determinada busca (KADHIM, 2019).

A primeira métrica extraída é a frequência do termo (TF – *Term Frequency*) em um determinado documento $TF(t_i,D)$ que tem a função de responder quão importante é uma palavra para um documento específico, conforme a Equação 1 (BUTTCHER; CLARKE; CORMACK, 2016). A métrica verifica quantas vezes uma determinada palavra se repete em um documento t_i,D e depois divide esse valor pelo total de palavras no documento $\sum t^i,D$.

$$TF(t_i,D) = \frac{t_i,D}{\sum t^i,D} \tag{1}$$

A segunda métrica extraída é a frequência inversa do documento (IDF - *Inverse Document Frequency*) $IDF(t_i)$ que responde quão importante uma palavra é entre todos os documentos da coleção, conforme a Equação 2 (ROBERTSON, 2004). Ela é calculada como o logaritmo do número de documentos no corpus (coleção de documentos) N dividido pelo número de documentos onde o termo específico aparece df. O cálculo do logaritmo é utilizado para normalização, para que palavras que são muitas vezes repetidas não tenham um peso muito alto.

$$IDF(t_i) = log\left(\frac{N}{df}\right) \tag{2}$$

A terceira métrica extraída é a frequência do termo inverso do documento (TF-IDF - *Term Frequency Inverse Document Frequency*) que descreve uma das maneiras pelas quais um mecanismo de pesquisa pode determinar se um texto é relevante em relação aos termos usados em uma consulta, conforme a Equação 3 (PENG; LIU; ZUO, 2014). Essa métrica é calculada multiplicando o valor do TF pelo IDF.

$$TF\text{-}IDF = TF(t_i,d) \cdot IDF(t_i) \tag{3}$$

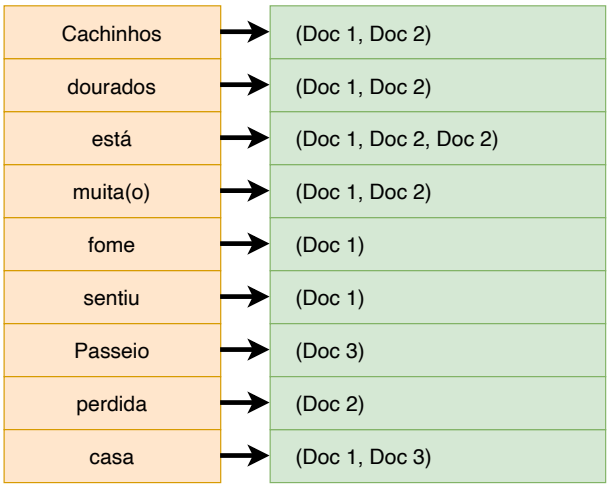
2.4.3 Índice Invertido

O índice invertido é uma estrutura fundamental e amplamente utilizada na recuperação de informação. Para cada palavra única que ocorre em uma coleção de documentos, o índice invertido armazena uma lista dos documentos em que essa palavra ocorre (PATIL *et al.*, 2011). A principal vantagem da utilização de um índice invertido é seu tamanho reduzido, pois, uma grande coleção de documentos de texto vai gerar um índice muito menor. Se forem utilizadas

técnicas de remoção de *stopwords* e *stemming*, o índice invertido pode ter um tamanho ainda menor (BAEZA-YATES; RIBEIRO-NETO, 2013).

Para que seja possível construir um índice invertido, primeiramente é necessário que o documento passe pelo pré-processamento, como visto na Figura 7. Quando o pré-processamento termina, é retornado para cada documento um vetor de palavras com as quais inicia-se o processo de construção do índice invertido, como pode-se observar na Figura 8. Primeiramente, são removidas todas as palavras repetidas contidas nos 3 documentos *(Doc1, Doc2, Doc3)*, depois cada palavra aponta para os documentos que possuem essa palavra.

Figura 8 – Exemplo de índice invertido.



Fonte: Adaptado de Baeza-Yates e Ribeiro-Neto (2013).

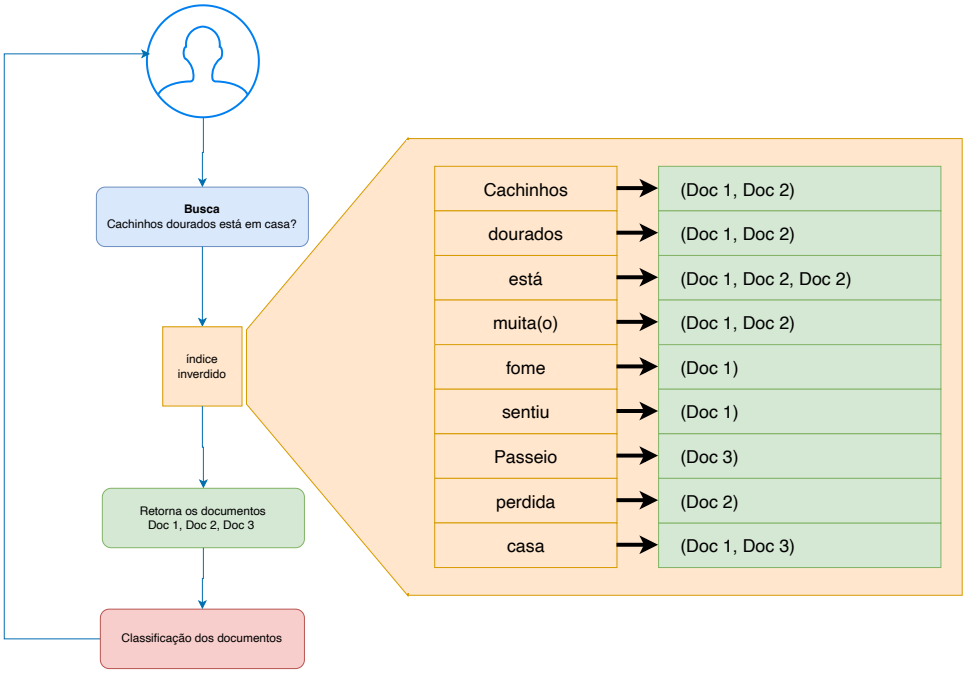
O uso do índice invertido traz vários benefícios. Primeiro, é possível manter todo o índice na memória RAM sem custos financeiros e computacionais elevados, pois o índice ocupa pouca memória. Segundo, as buscas provêm um melhor desempenho, pelo fato de o índice estar na memória e não ser necessário acessar cada documento durante a busca. Todo processo pode ser feito acessando somente o índice invertido (MANNING, 2008).

2.4.4 Busca e Classificação

A última etapa em sistemas de recuperação da informação é a busca, pois esta só é possível após a criação de um índice invertido. O modelo de busca de um por vez é a maneira mais simples de encontrar documentos dentro de um índice invertido (PATIL *et al.*, 2011).

Como pode-se observar na Figura 9, o usuário executa uma busca em linguagem natural *"cachinho dourados está em casa"* e o sistema de recuperação de informação submete cada palavra da busca ao índice invertido. O índice invertido retornará os documentos que contêm as palavras da busca. Após o retorno dos documentos pelo índice invertido, estes são submetidos a uma função de classificação que tem como objetivo escolher quais são os documentos que devem aparecer nas primeiras posições, ou seja, quais são os documentos mais relevantes para o usuário. Feita a classificação dos documentos, eles são retornados para os usuários.

Figura 9 – Etapa de busca e classificação de documentos.



Fonte: Autoria própria.

No campo de sistemas de recuperação de informações, existem diversas métricas para avaliar a relevância de um documento, que são (BAEZA-YATES; RIBEIRO-NETO, 2013):

- Similaridade Jaccard: esta métrica é determinada pelo número de termos compartilhados entre dois documentos. Uma maior quantidade de termos em comum indica uma maior similaridade entre os documentos;
- Similaridade Dice: semelhante à similaridade Jaccard, essa métrica também considera o número total de termos presentes em cada documento para calcular a similaridade. A principal diferença é que essa métrica dá mais peso ao número de termos que os dois documentos têm em comum;
- KL-divergência: Baseada na diferença entre as distribuições de frequência de termos nos documentos comparados, essa métrica sugere que quanto maior a diferença entre as distribuições, menor é a similaridade entre os documentos;
- Similaridade do cosseno: é uma métrica de similaridade entre vetores. Os vetores de termos são usados para representar documentos. A similaridade do cosseno é calculada como o produto escalar entre os vetores de termos dos dois documentos. Um produto escalar maior indica uma maior similaridade entre os documento;
- BM25: é uma métrica de similaridade frequentemente utilizada na recuperação de informações. A métrica BM25 considera o número de termos comuns aos documentos, o comprimento dos documentos e a frequência dos termos nos documentos.

Em geral, os sistemas de recuperação de informações em buscas clássicas se baseiam em palavras-chave ou termos diretamente relacionados ao léxico dos documentos (MANNING, 2008). Entre os modelos existentes, o BM25 se destaca em termos de desempenho por várias razões. Primeiramente, ele é um modelo de ponderação de termos que leva em conta não só a frequência de um termo em um documento específico, mas também a frequência do termo em todo o conjunto de documentos e o comprimento do documento. Essas características permitem que o BM25 represente o significado de um documento de forma mais precisa quando comparado a modelos de similaridade mais simples, como a similaridade de cosseno (BAEZA-YATES; RIBEIRO-NETO, 2013).

O BM25 é um método de ranqueamento popular na recuperação da informação. Ele é um modelo probabilístico que classifica um documento D com base na probabilidade deste corresponder a uma determinada busca Q, conforme a Equação 4 (SHI; KEUNG; SONG, 2014).

$$\text{BM25}(D,Q) = \sum_{i=1}^{n} \text{IDF}(t_i) \cdot \frac{TF(t_i,D) \cdot (k_1 + 1)}{TF(t_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})} \tag{4}$$

Cada termo (palavra) t_i do documento D (t_i, D) da busca Q possui uma pontuação que depende da ocorrência desse termo em todos os documentos, independentemente da inter-relação entre os termos de consulta dentro de um documento (SHI; KEUNG; SONG, 2014). A constante k_1 ajuda a limitar o quanto um único termo de consulta pode afetar a pontuação de um determinado documento e a constante b serve para controlar a influência do tamanho do documento, $|D|$ é o número de palavras contidas no documento D e avgdl é a média de palavras existentes em todos os documentos da coleção (CONNELLY, 2018). O resultado da classificação indica que quanto mais termos no documento corresponderem à busca, maior deve ser a pontuação deste documento.

Um documento com um valor maior de BM25 é considerado mais relevante. Em seguida, os documentos recuperados são retornados para os usuários em ordem decrescente de relevância, de acordo com os resultados do algoritmo de classificação (MANNING, 2008).

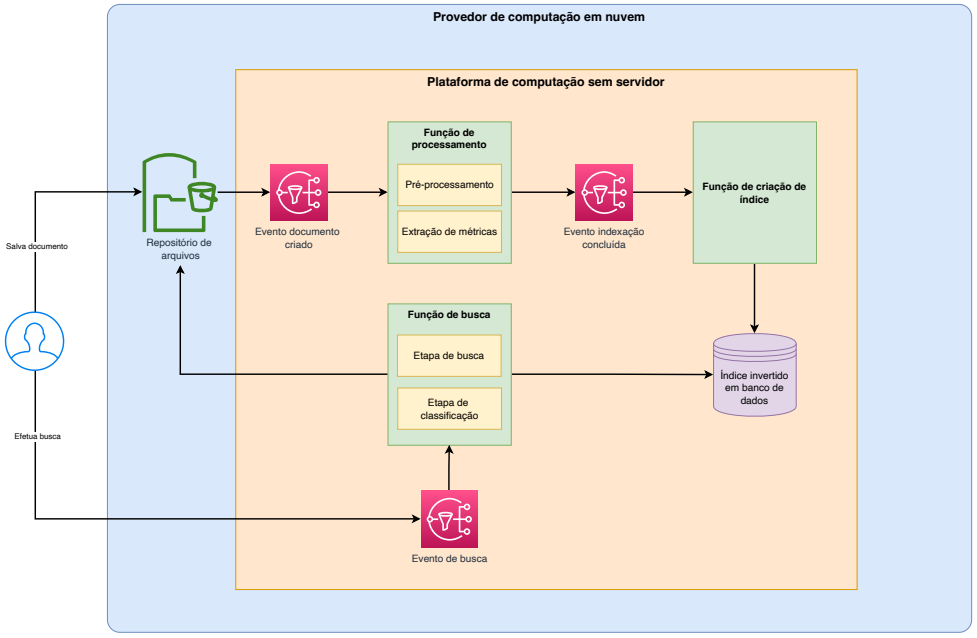
2.4.5 Discussões

Essa seção tem como objetivo demostrar como os assuntos apresentados nesse capítulo se relaciona. Para isso, criou-se uma arquitetura genérica de um sistema de recuperação da informação com computação sem servidor, conforme ilustra a Figura 10.

Na arquitetura genérica, antes que as buscas possam ser efetuadas, é necessário adicionar os documentos de texto em um repositório de documentos, para então iniciar o processo de indexação. Caso contrário, a busca não retornaria nenhum documento.

Sempre que um documento é adicionado em um repositório de arquivos, ele gera um evento assíncrono de documento criado. Quando o evento ocorre, a função de processamento (software executado sobre a plataforma de computação sem servidor) recebe o arquivo e exe-

Figura 10 – Exemplo de arquitetura genérica utilizando computação sem servidor.



Fonte: Autoria própria.

cuta dois processos importantes que é, o pré-processamento (Seção 2.4.2.1) e a extração de métricas (Seção 2.4.2.2). Após realizada a função de processamento, gera-se um novo evento assíncrono de pré-processamento concluído.

A função de criação de índice reage ao evento e recebe as informações de pré-processamento e métricas do texto. Com isso, inicia a criação do índice invertido (Seção 2.4.3). O índice invertido é persistido em um banco de dados para que seja possível realizar consultas posteriormente.

Agora que os documentos foram indexados, o usuário pode executar buscas. O usuário gera um evento de busca síncrono, que faz com que a função de busca seja executada. A função de busca primeiramente identificará quais documentos correspondem às palavras da busca. Após isso, inicia-se a etapa de classificação, que definirá quais são os documentos mais relevantes. Essas etapas foram discutidas na Seção 2.4.4. Por último, a função de busca acessa o repositório de documentos para obter os documentos retornados para o usuário, por ordem de classificação mais alta.

2.4.6 Métricas de Avaliação em Sistemas de Recuperação da Informação

Avaliar um sistema de recuperação da informação (RI) é importante para determinar se ele está atendendo às necessidades dos usuários em relação as suas buscas. Existe duas categorias de avaliação: avaliação do sistema e avaliação do usuário. A avaliação do usuário mede a sua satisfação com o sistema, enquanto a avaliação do sistema se concentra na qualidade da classificação de documentos. Como a avaliação do usuário é muito mais cara e

difícil de ser feita corretamente, os pesquisadores de RI costumam usar a avaliação do sistema (VOORHEES, 2002).

A avaliação de um sistema de RI foi feita pela primeira vez com o experimento chamado *Cranfield*. O experimento introduziu um paradigma para avaliação de sistemas que tem sido o modelo dominante há quatro décadas. Esse modelo é usado pela conferência de recuperação de texto (TREC - *Text Retrieval Conference*) que foi fundada em 1992 pelo Instituto Nacional de Padrões e Tecnologia (NIST - *National Institute of Standards and Technology*) dos EUA (VOORHEES, 2002). O objetivo do TREC foi construir uma coleção de testes de grande porte para pesquisa em recuperação de informação. O TREC alcançou esse objetivo e, juntamente com outras conferências de avaliação, construiu outras dezenas de coleções de testes para várias tarefas. Essas conferências também estabeleceram e validaram melhores práticas para o uso de coleções de testes (VOORHEES, 2019).

Uma coleção de teste é composta pelos seguintes componentes (SANDERSON *et al.*, 2010):

- Uma coleção de documentos utilizada para ser indexada em um sistema de RI. Cada documento recebe um único identificador *docid*;
- Um conjunto de consultas escrita em linguagem natural. Cada consulta possui um identificador chamado *qid*;
- Um conjunto de julgamentos de relevâncias (qrels - *query relevance set*), composto por uma lista tripla contendo o identificador da consulta (*qid*), o identificador do documento indexado (*docid*) e qual o nível de relevância do documento retornado para a consulta executada.

Um teste de um sistema de RI é executado em duas partes. Primeiro, os documentos da coleção de teste são indexados no sistema de RI com seus respectivos identificadores (docids). Em seguida, é realizado um conjunto pré-determinado de consultas. Com o resultado dessas consultas, é gerada uma lista de documentos recuperados, que é comparada com as qrels (lista de documentos relevantes para cada consulta). Na segunda parte, aplicam-se métricas de avaliação (SANDERSON *et al.*, 2010).

Cleverdon (1966) identificou uma lista de seis métricas para avaliar um sistema de RI, incluindo *recall*, precisão (*precision*), cobertura, tempo de resposta, questões de apresentação e esforço do usuário. Embora haja muitos experimentos que abrangem diferentes aspectos dessa lista, os principais critérios de avaliação são *recall* e precisão, pois se concentram na eficácia da recuperação (DUNLOP, 1997).

A métrica de *recall* é importante porque ela indica a capacidade do sistema de RI de encontrar todos os documentos relevantes para uma determinada consulta (CHOWDHURY, 2010). A Equação 5 exemplifica o cálculo da métrica *recall*, onde divide-se o número de documentos

relevantes que foram recuperados (R) pelo número total de documentos relevantes disponíveis para aquela busca específica na coleção de teste (C) (BUTTCHER; CLARKE; CORMACK, 2016). Um sistema de RI com alta *recall* é capaz de encontrar a maioria dos documentos relevantes, enquanto um sistema com baixo *recall* pode perder documentos relevantes importantes.

$$\text{Recall} = \frac{R}{C} \tag{5}$$

Suponha que a coleção de teste contenha 30 documentos relevantes para uma determinada consulta (C) e que o sistema retornou 20 documentos, mas somente 10 deles são relevantes (R). Isso significa que o sistema de RI tem uma taxa de *recall* de 33% ($R = \frac{10}{30}$) dos documentos relevantes disponíveis na coleção de teste para essa busca específica.

A métrica de precisão é importante porque ela indica a capacidade do sistema de RI retornar apenas documentos relevantes para uma determinada consulta. A Equação 6 exemplifica o cálculo da métrica de precisão, onde divide-se a quantidade de documentos relevantes que foram recuperados (R) pela quantidade total de documentos retornados pela busca (L) (BUTTCHER; CLARKE; CORMACK, 2016).

$$\text{Precisão} = \frac{R}{L} \tag{6}$$

Suponha que, após uma busca, o sistema de RI retorne 20 documentos relacionados à palavra-chave da busca (L). Porém, desses 20 documentos, apenas 10 são considerados relevantes de acordo com a coleção de teste para essa consulta (R). Isso resulta em uma taxa de precisão de 0,5 ($R = \frac{10}{20}$), indicando que 50% do resultado da busca é relevante para o usuário.

Com os resultados das métricas *recall* e precisão, é possível gerar a média harmônica, também conhecida como F1 (ver Equação 7), que é útil quando ambas as métricas são importantes para a avaliação de um sistema de RI (BUTTCHER; CLARKE; CORMACK, 2016).

$$F1 = 2 \cdot \left(\frac{\text{Precisão} \cdot \text{Recall}}{\text{Precisão} + \text{Recall}} \right) \tag{7}$$

Por exemplo, em algumas aplicações é preciso ter uma alta taxa de *recall*, enquanto em outras, a taxa de precisão é mais importante. A média harmônica F1 permite avaliar o desempenho de um sistema de RI levando em conta esses dois fatores simultaneamente.

A avaliação utilizando a precisão e *recall* é importante, pois consegue avaliar se o sistema é capaz de retornar documentos relevantes. Porém, não consegue avaliar se os documentos mais relevantes para os usuários estão nas primeiras posições. Para esse objetivo é utilizada a avaliação de ganho cumulativo descontado normalizado (NDCG - *Normalized Discounted Cumulative Gain*).

O NDCG tem duas vantagens em comparação com muitas outras métricas. Primeiro, o NDCG permite que cada documento recuperado tenha um grau de relevância, enquanto a

maioria das métricas de classificação tradicionais permitem apenas relevância binária. Isto é, cada documento é visto como relevante ou não relevante por métricas de classificação como precisão e *recall* (WANG *et al.*, 2013). A segunda vantagem é que o NDCG envolve uma função de penalidade sobre a classificação, enquanto muitas outras métricas pesam uniformemente todas as posições (WANG *et al.*, 2013). Essa característica é particularmente importante para os mecanismos de pesquisa, pois os usuários se importam muito mais com os documentos mais relevantes nas primeiras posições.

Para fazer uma avaliação utilizando o NDCG primeiramente é preciso calcular o ganho acumulativo descontado (DCG - *Discounted Cumulative Gain*) e o ganho acumulativo descontado ideal (IDCG - *Ideal Discounted Cumulative Gain*).

O DCG é uma métrica de avaliação de sistemas de RI que mede a qualidade relativa de uma lista de resultados da busca. A Equação 8 mostra o cálculo da métrica DCG, a qual realiza a soma ponderada dos ganhos de relevância dos resultados $rel(r_1) + \frac{rel(r_2)}{\log_2(2)} + \cdots + \frac{rel(r_n)}{\log_2(n)}$, onde a relevância de cada resultado $rel(r_n)$ é ponderada pela sua posição na lista de resultados n (BUTTCHER; CLARKE; CORMACK, 2016). A ponderação é fornecida pela função de penalidade, que é uma função logarítmica $\log_2(n)$.

$$DCG = rel(r_1) + \frac{rel(r_2)}{\log_2(2)} + \cdots + \frac{rel(r_n)}{\log_2(n)} \tag{8}$$

Uma coleção de teste possui, para cada busca, uma lista de documentos (docid) contendo em qual posição eles devem aparecer para uma determinada busca e qual é a relevância do documento naquela posição. Suponha que seja executada uma busca em um sistema de IR e ela retorne 5 documentos. Para avaliar a qualidade dos resultados, cada documento retornado é comparado com as informações fornecidas pelos qrels. Dessa forma, é possível determinar o grau de relevância de cada documento em relação à consulta executada. Para exemplificar, a Equação 9 mostra o cálculo da métrica de DCG considerando o grau de relevância 2, 4, 5, 0 e 2 para os documentos de 1 a 5, respectivamente.

$$2 + \frac{4}{\log_2(2)} + \frac{5}{\log_2(3)} + \frac{0}{\log_2(4)} + \frac{2}{\log_2(5)} = 10.01 \tag{9}$$

O ganho acumulativo descontado ideal (IDCG - *Ideal Discounted Cumulative Gain*) utiliza a função do DCG para o cálculo, mas são passadas as informações da coleção de teste e não o resultado da busca no sistema de RI. O IDCG, Equação 10 é o valor máximo possível de DCG para uma determinada consulta. Ele é calculado considerando a ordem ideal de classificação dos documentos (BUTTCHER; CLARKE; CORMACK, 2016).

Digamos que o grau de relevância contida na coleção de teste seja 5, 4, 3, 2 e 1 para os documentos de 1 a 5, respectivamente. Então, o IDCG é calculado conforme a Equação 10 (BUTTCHER; CLARKE; CORMACK, 2016).

$$5 + \frac{4}{\log_2(2)} + \frac{3}{\log_2(3)} + \frac{2}{\log_2(4)} + \frac{1}{\log_2(5)} = 12.02 \quad (10)$$

Com os resultados do IDCG e do DCG é possível calcular o NDCG, conforme a Equação 11 (BUTTCHER; CLARKE; CORMACK, 2016).

$$NDCG = \frac{DCG}{IDCG} \quad (11)$$

A avaliação via NDCG é importante porque ele normaliza o DCG e permite comparar o desempenho de diferentes sistemas de RI, independentemente da quantidade e da relevância dos documentos retornados. O resultado do NDCG varia entre 0 e 1, onde 1 é o melhor valor possível (BUTTCHER; CLARKE; CORMACK, 2016).

2.4.7 Apache Lucene e Elastic Search

Apache Lucene é uma biblioteca de pesquisa moderna e de código aberto construída em Java (BIAŁECKI *et al.*, 2012). A biblioteca foi projetada para entregar resultados relevantes com alto desempenho para sistemas de RI. Empresas como Twitter, Netflix e Instagram utilizam o Lucene em suas buscas (BIAŁECKI *et al.*, 2012). O Apache Lucene fornece interfaces de programação para processamento de linguagem, criação de índices, busca e classificação.

Para indexação dos documentos, o Lucene executa três tarefas que são encadeadas (BIAŁECKI *et al.*, 2012). Primeiramente, são removidos acentos e caracteres especiais. Depois, executa-se a etapa de tokenização. Por fim, são removidos os *stopwords*, *stemming* e gera-se o vetor de palavras. De posse do vetor de palavras, constrói-se um índice invertido.

O Lucene contém uma grande coleção de consultas predefinidas. Com isso, os desenvolvedores podem expressar critérios complexos para correspondência e pontuação de documentos (BIAŁECKI *et al.*, 2012). Quando uma consulta é executada, cada segmento de índice invertido é processado sequencialmente. São retornados os documentos que correspondem às buscas e na sequência classifica-se o documento mais relevante para o usuário.

O *Elastic Search* (ELS) é um sistema de RI distribuído que utiliza o Apache Lucene, de modo a armazenar, pesquisar e analisar grandes volumes de dados não estruturados em tempo real. Seu mecanismo de busca e análise de texto permite uma ampla gama de recursos avançados, como pesquisa de texto completo, suporte a várias linguagens, agregação de dados e análise de dados geoespaciais (GORMLEY; TONG, 2015).

Com a capacidade de dimensionar horizontalmente, o *Elastic Search* pode lidar com grandes quantidades de dados e tráfego de usuários. Devido a essas capacidades, ele é amplamente utilizado por empresas em uma ampla gama de casos de uso, incluindo busca em aplicação, busca em *website*, busca empresarial e *logging* (GORMLEY; TONG, 2015).

3 TRABALHOS RELACIONADOS

Esse capítulo apresenta o estado da arte relacionado ao uso da computação sem servidor e principalmente em casos que envolvem recuperação da informação.

A computação sem servidor está ganhando espaço em várias áreas de redes de computadores, incluindo Redes Definidas por Software (SDN - *Software Defined Networking*) (KRÓL; PSARAS, 2017). O estudo de Aditya *et al.* (2019) abordou uma maneira de fazer com que as redes sejam mais fáceis de gerenciar e adaptar. Neste modelo, a rede é dividida em duas partes: uma que controla e a outra que encaminha os dados. Esse desacoplamento torna os comutadores de rede mais simples, já que eles só precisam encaminhar os dados. Os elementos de controle, por outro lado, podem ser programas separados que funcionam em plataformas sem servidor. Por exemplo, quando um pacote de dados chega a um comutador de rede SDN, o cabeçalho do pacote é analisado e a informação é enviada para o controlador SDN. O controlador, então, decide o que fazer com o pacote e envia essa decisão de volta ao comutador para que ele execute a ação.

A integração entre computação sem servidor e computação de ponta tem possibilitado o desenvolvimento de aplicações diversas e inovadoras. Em um estudo realizado por Tran *et al.* (2020), foi concebido um sistema para a operação de robôs móveis autônomos (AMRs - *Autonomous Mobile Robots*) utilizando ambas as tecnologias. Este sistema é composto por três elementos-chave: um robô móvel autônomo para computação de ponta, a plataforma de computação sem servidor da AWS e um aplicativo móvel de controle. O objetivo principal deste sistema é automatizar o processo de entrega de pacotes. Na prática, o usuário interage com o aplicativo móvel para solicitar uma entrega de pacote. Ao receber a solicitação, o serviço de IoT da AWS dispara funções Lambda que determinam as coordenadas para a entrega. Em seguida, o robô móvel autônomo é acionado para executar a tarefa de entrega.

Em Crane e Lin (2017) foi proposta a primeira aplicação de recuperação da informação sobre a plataforma de computação sem servidor (Serverless). Para implementação do projeto, foi utilizado o serviço de computação em nuvem da AWS (*Amazon Web Serverces*). O projeto proposto aborda a etapa busca e classificação. Apesar do sistema possuir um índice, toda etapa de indexação de documentos não utilizou a computação sem servidor. Esse trabalhou conseguiu demostrar que é possível a implementação de uma solução de recuperação da informação sobre computação sem servidor.

Em Lin (2020) foi realizado um experimento que adapta o *Apache Lucene* para ser utilizado com computação sem servidores. O *Apache Lucene* é uma biblioteca de código aberto que ajuda na construção de sistemas de recuperação da informação. Porém, a proposta tem a limitação de assumir índices estáticos, ou seja, não existe nenhum mecanismo de indexação de documentos. Outra limitação é que o projeto atual assume que todo o índice caberá em uma única instância *Lambda* com recursos de memória limitados.

Em Anand *et al.* (2021) foi desenvolvido um protótipo contemplando a etapa de busca de um sistema de recuperação da informação sobre a computação sem servidor. Os autores, usaram dois métodos de classificação de documentos. A primeira etapa da recuperação é feita utilizando *Apache Lucene* com um algoritmo de classificação probabilístico BM25 e depois os documentos são reordenados utilizando o modelo *MonoBERT* utilizando a arquitetura de transformadores (*transformers*). Os índices invertidos são construídos localmente e logo após são carregados para o repositório de arquivos na nuvem (S3 - *Amazon Simple Storage Service*). O protótipo tem problemas com latência, principalmente pelo uso da arquitetura de transformadores e enfrenta problemas com os custos financeiros excessivos.

Em Baresi, Mendonça e Garriga (2017) foi proposta uma arquitetura de computação sem servidor para computação em borda (*Edge Computing*). O objetivo é permitir que aplicativos móveis acessem serviços com baixa latência, com isso economizando recurso de bateria e processador. Para avaliar a arquitetura proposta, foi utilizado um aplicativo de realidade aumentada com técnicas de recuperação da informação. A arquitetura proposta superou a nuvem em até 80% em termos de taxa de transferência e latência.

As pesquisas encontradas na literatura vêm se concentrando em propostas de arquitetura para sistema de recuperação da informação, que possa vencer nas particularidades da computação sem servidor, como, baixa memória e tempo limitado de processamento. Poucos trabalhos foram publicados até o momento e nenhum trabalho aborda as técnicas da etapa de indexação, focando apenas nas etapas de busca e classificação.

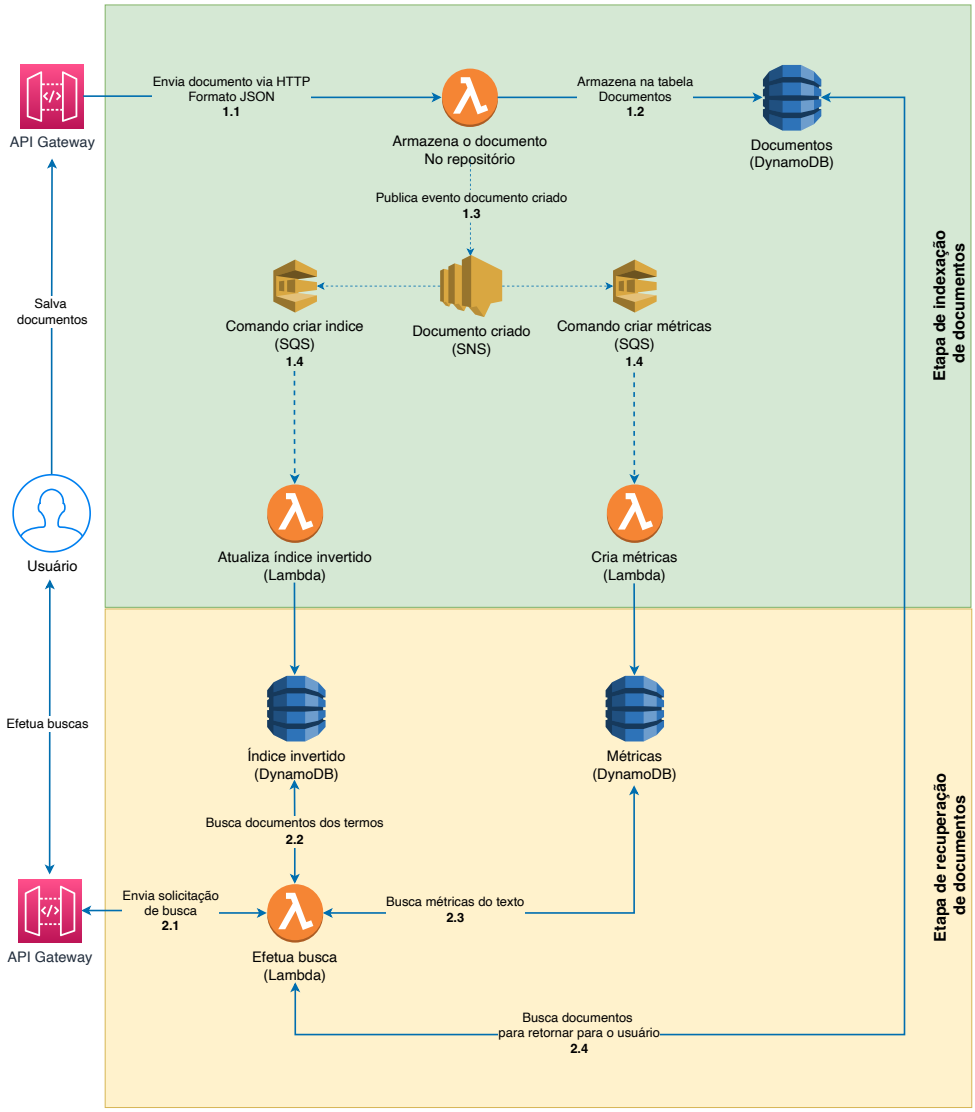
As principais contribuições desse trabalho é dar continuidade às pesquisas realizadas envolvendo o uso da computação sem servidor na recuperação da informação e propor uma arquitetura de sistema de recuperação da informação que aborde as etapas de indexação, busca e classificação.

4 ARQUITETURA SIRA

Esse capítulo apresenta a arquitetura denominada SIRA (*Serverless Information Retrieval Architecture*) para a recuperação da informação sobre a plataforma de computação sem servidor.

A arquitetura SIRA é dividida em duas etapas, conforme pode ser visto na Figura 11. A primeira etapa é responsável pela indexação dos documentos, tendo a finalidade de receber um documento, logo em seguida persisti-lo, extrair métricas do texto e criar um índice invertido. A segunda parte é responsável pela recuperação dos documentos, que vai receber uma busca escrita em linguagem natural, buscar documentos no índice invertido que contenham os termos da busca, obter informações de métricas sobre esses documentos, com as métricas classificar os melhores documentos e, por último, mostrar o resultado para o usuário. Os detalhes da arquitetura são apresentados nas Seções 4.1 e 4.2

Figura 11 – Arquitetura SIRA.



Fonte: Autoria própria.

A arquitetura proposta utiliza a plataforma de computação sem servidor da AWS, conhecida como Lambda. Essa plataforma permite a execução de funções, seguindo o padrão de microsserviços, o que garante escalabilidade e flexibilidade na execução dos serviços.

Todo acesso aos recursos vão passar por um *API Gateway*. A exposição de contratos de serviços através de um *API Gateway* é utilizada para manter um acoplamento fraco entre o consumidor e o serviço.

Para o componente de banco de dados foi utilizado o *DynamoDB*. *DynamoDB* é um servidor de banco de dados distribuído disponibilizado como serviço pela AWS. Esse serviço foi escolhido pelo seu alto nível de tolerância a falhas. Para executar aplicações com alta escalabilidade, é necessário que serviços de dados possam ser dimensionados para operar em milhares de servidores e lidar com vários tipos de falhas, como falhas de servidor e partições de rede.

Foi utilizado o SNS (*Simple Notification Service*) como serviço de mensagens no modelo *publisher/subscriber* para mensagens do tipo eventos. Para mensagens do tipo comando, foi utilizado o SQS (*Simple Queue Service*), que trabalha no modelo de filas. Com a utilização de eventos assíncronos na arquitetura, é possível alcançar maior escalabilidade, disponibilidade e isolamento de falhas.

As etapas de indexação e recuperação de documentos são detalhadas nas próximas seções.

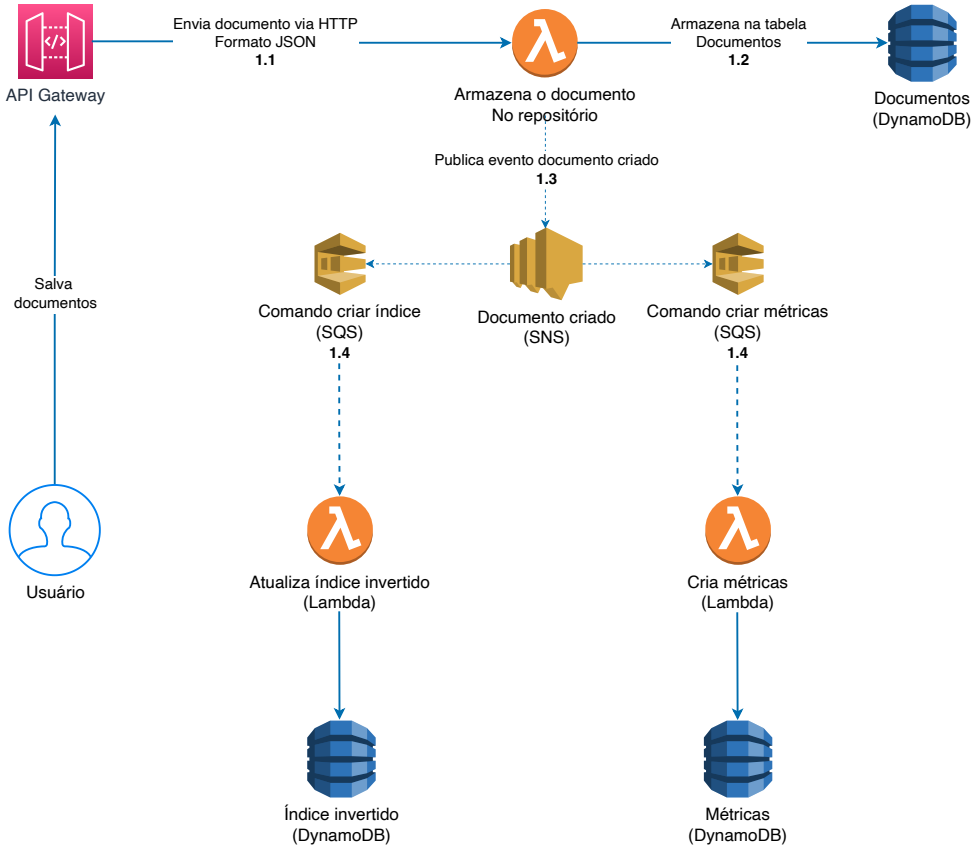
4.1 Indexação de Documentos

Na arquitetura SIRA, a etapa de indexação do documento, ilustrada na Figura 12, acontece nos passos 1.1, 1.2, 1.3 e 1.4. No passo 1.4 acontece duas etapas de forma paralela: o comando criar índice e o comando criar métricas.

No processo de indexação de documentos, um usuário envia documentos de texto para um *API Gateway* através uma requisição HTTP de tipo POST. O API *API Gateway*, no passo 1.1, gerará um evento síncrono que executará o serviço *"armazenamento do documento"*. O corpo desse evento conterá o documento no formato JSON. O serviço *"armazenamento do documento"* recebe somente um documento por requisição, para que seu processamento seja rápido. Essa abordagem foi utilizada na arquitetura SIRA, pois na computação sem servidor quanto maior o tempo de execução do processo, maior são os custos. Então, é salvo o documento em uma base de dados para recuperação posterior, como é possível ver no passo 1.2. Uma vez armazenado, no passo 1.3, é gerado um evento com o documento que acabou de ser salvo no corpo da mensagem. Nos passos 1.4 foi utilizada a técnica de *fanout* para distribuir uma mensagem para vários destinos simultaneamente. O SNS distribui a mensagem para cada fila SQS, permitindo que a mensagem seja tratada de forma paralela por dois serviços diferentes ("Atualiza índice invertido" e "Cria métricas").

O serviço *"Atualiza índice invertido"* é responsável por receber um documento e prepará-lo para ser adicionado ao índice invertido (Seção 2.4.3). Isso é alcançado através de uma etapa

Figura 12 – Processo de indexação de documentos.



Fonte: Autoria própria.

de pré-processamento (Seção 2.4.2.1), onde são removidos caracteres especiais, como mostrado no Algoritmo 1. Primeiramente é recebido um documento. Esse documento tem duas propriedades, *"id"* contendo um código único para o documento e o *"texto"* do documento (Linha 1 e 2). Então, é criado um vetor de palavras a partir de um texto, utilizando a função *"separarTextoEmPalavras"* (Linha 3). Em seguida, é criado um vetor de códigos ASCII que armazena os códigos dos caracteres que devem ser removidos do texto (Linha 4). Na linha 5, é criada uma variável chamada *"vetorPalavrasNormalizadas"*, que tem a função de armazenar as palavras sem caracteres especiais. O algoritmo percorre cada palavra do vetor de palavras e, logo em seguida, cria uma variável chamada *"vetorPalavraNormalizada"* (Linhas 6 e 7), que armazenará a palavra sem os caracteres especiais. Então, o algoritmo percorre cada caractere de uma determinada palavra e verifica se seu código ASCII está contido no *"vetorCodigosAscii"* (Linha 9 e 10). Se não estiver contido, significa que é um caractere válido e, então, o algoritmo adiciona o caractere ao *"vetorPalavraNormalizada"* (Linha 11). Caso contrário, o caractere é ignorado. Após percorrer todos os caracteres da palavra, o algoritmo adiciona o *"vetorPalavraNormalizada"*, que contém a palavra sem os caracteres especiais, ao *"vetorPalavrasNormalizadas"* (Linha 14). O serviço *"atualiza índice invertido"* também realiza a remoção de palavras de baixa qualidade semântica, conhecidas como *stopwords*. Na linha 16 é criada uma lista de *"stopwords"*. Então, percorre-se cada palavra do *"vetorPalavrasNormalizadas"* e é verificado se

a palavra está contida na lista de *"stopwords"* (Linha 17 a 20). Se a palavra estiver contida na lista, ela é removida do *"vetorPalavrasNormalizadas"* (Linha 22). Na linha 22 inicia-se um dicionário chamado *"indiceInvertido"*. Em seguida, o algoritmo percorre cada palavra do vetor de palavras normalizadas (Linha 23), verificando se a palavra já está presente no *"indiceInvertido"* (Linha 24). Se a palavra não estiver presente, é criado um novo vetor chamado *"vetorDocumentos"* (Linha 25), adiciona-se o identificador do documento a esse vetor (Linha 26) e associa-se o vetor *"vetorDocumentos"* à palavra no *"indiceInvertido"* (Linha 27). Se a palavra já estiver presente no *"indiceInvertido"*,recupera-se o vetor de documentos associado à palavra e adiciona-se o identificador do documento a esse vetor (Linha 29 a 30). Ao final da iteração por todas as palavras, o *"indiceInvertido"* terá uma entrada para cada palavra, onde o valor associado a essa entrada é um vetor contendo os identificadores dos documentos em que a palavra aparece.

Algoritmo 1 – Algoritmo de pré-processamento e criação do índice invertido.

```
 1: idDocumento ← documento.id
 2: texto ← documento.texto
 3: vetorPalavras ← separarTextoEmPalavras(texto)
 4: vetorCodigosAscii ← [35,36,37,48,57,65,90,97,122]
 5: vetorPalavrasNormalizadas ← []
 6: para cada palavra no vetorPalavras faça
 7:     vetorPalavraNormalizada ← []
 8:     para cada caractere da palavra faça
 9:         codigoAsciiDoCaractere ← obtemCódigoAsciiDoCaractere(caractere)
10:         se codigoAsciiDoCaractere não existe no vetorCodigosAscii então
11:             vetorPalavraNormalizada.adiciona(caractere)
12:         finaliza se
13:     finaliza para
14:     vetorPalavrasNormalizadas.adiciona(vetorPalavraNormalizada)
15: finaliza para
16: stopwords ← ["a", "o", "que", "de", "e", "do", "da", "em", "um", "para"]
17: para cada palavra no vetorPalavrasNormalizadas faça
18:     se palavra existe em stopWords então
19:         vetorPalavrasNormalizadas.remove(palavra)
20:     finaliza se
21: finaliza para
22: indiceInvertido ← {}
23: para cada palavra do vetorPalavrasNormalizadas faça
24:     se palavra não está em indiceInvertido então
25:         vetorDocumentos ← []
26:         vetorDocumentos.adicionar(idDocumento)
27:         indiceInvertido[palavraNormalizada] ← vetorDocumentos
28:     senão,
29:         vetorDocumentos ← indiceInvertido[palavraNormalizada]
30:         vetorDocumentos.adicionar(idDocumento)
31:     finaliza se
32: finaliza para
```

Fonte: Autoria própria.

Após o serviço *"atualiza índice invertido"* ser completado, o índice invertido é atualizado no banco de dados DynamoDB. O índice invertido é organizado de forma simples em uma estrutura de dados do tipo dicionário, onde cada termo é associado a uma lista de documentos onde ele aparece. Dessa forma, quando uma consulta é realizada, o sistema de busca pode rapidamente encontrar os documentos que contêm os termos da busca, sem precisar percorrer toda a coleção.

O serviço *"Cria métricas"* vai receber o documento, esse documento vai passar pela etapa de pré-processamento. O resultado do processo é um vetor de palavras. Através desse vetor são geradas as métricas de frequência do termo no documento (TF) (Seção 2.4.2.2). Esse processo é ilustrado no Algoritmo 2. O processo é iniciado com a inicialização do vetor de palavras, que armazenará as palavras extraídas do documento após o pré-processamento (Linha 1), e um dicionário de frequências de termos, que armazenará a contagem de cada palavra encontrada no documento (Linha 2). Em seguida, o vetor de palavras é preenchido com os resultados do pré-processamento do documento (Linha 3). O pré-processamento inclui etapas como remoção de *stopwords* e normalização de texto. Após o preenchimento do vetor de palavras, o algoritmo entra em uma estrutura de laço que varre cada palavra do vetor e verifica se a palavra já está presente no dicionário de frequências de termos (Linha 4 e 5). Se a palavra estiver presente, é somado o valor anterior para a palavra mais um (Linha 6). Se não estiver presente, o dicionário *"dicionárioFrequênciasDoTermoTF"* é atualizado com a adição da nova palavra e o valor para ela é iniciado em 1 (Linha 8). Após o término do laço, o dicionário de frequências de termos conterá o valor de TF para cada palavra encontrada no documento.

Algoritmo 2 – Algoritmo de geração do TF (frequência do termo).

```
 1: vetorPalavras ← []
 2: dicionárioFrequênciasDoTermoTF ← {}
 3: vetorPalavras ← efetuaPréProcessamento(documento)
 4: para cada palavra no vetorPalavras faça
 5:     se palavra está contida na dicionárioFrequênciasDoTermoTF então
 6:         dicionárioFrequênciasDoTermoTF[palavra] ← dicionárioFrequênciasDoTermoTF[palavra] + 1
 7:     senão,
 8:         dicionárioFrequênciasDoTermoTF[palavra] ← 1
 9:     finaliza se
10: finaliza para
```

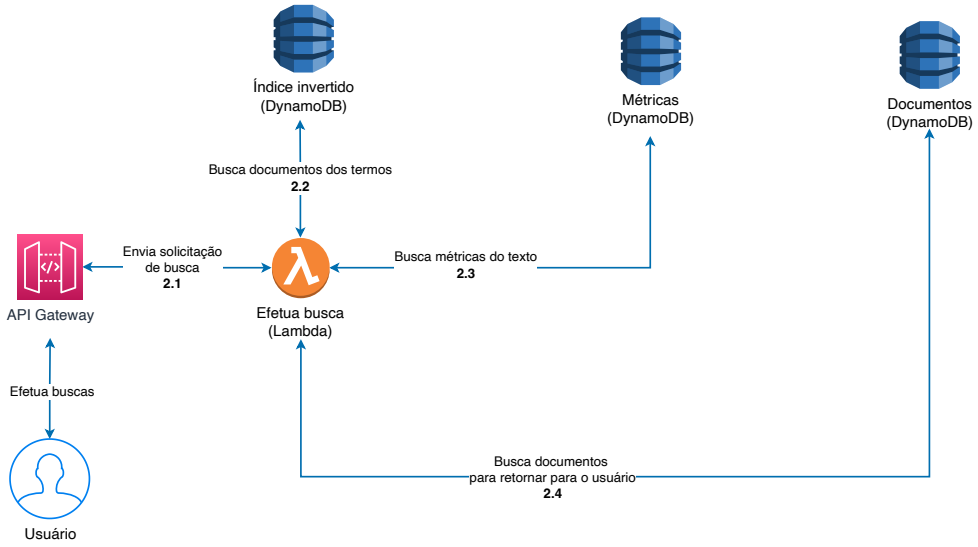
Fonte: Autoria própria.

Essas informações são persistidas no banco de dados de métricas e serão utilizadas para a classificação de documentos na etapa de recuperação de documentos que está detalhada na próxima seção.

4.2 Recuperação de Documentos

A Figura 13 ilustra o processo de recuperação de documentos. O usuário envia uma solicitação de busca que é processada pelo *API Gateway*. O *API Gateway* recebe uma requisição HTTP do tipo GET contendo os termos da busca escritos em linguagem natural. Em seguida, no passo 2.1, é gerado um evento síncrono que executa o serviço *"efetua busca"*.

Figura 13 – Processo de recuperação de documentos.



Fonte: Autoria própria.

O serviço *"Efetua Busca"* é responsável por processar as solicitações de busca dos usuários, encontrando os documentos mais relevantes para a consulta. Para isso, o serviço segue uma série de etapas. Primeiro é executado o passo 2.2 que está detalhado no Algoritmo 3. As variáveis *"documentos"* e *"vetorPalavras"* são inicializadas. A variável *"documentos"* é iniciada como um vetor vazio (Linha 1) e a variável *"vetorPalavras"* é iniciada com o resultado da função *"efetuaPréProcessamento"*, que recebe como entrada a busca feita pelo usuário (Linha 2). A segunda parte é o coração do algoritmo, o qual consiste em um laço que itera por cada palavra no vetor *"vetorPalavras"*(Linha 3). Dentro deste laço, é realizada uma condição para verificar se a palavra atual está presente no índice invertido (Linha 4). Se esta condição for verdadeira, a função *"acrescentar"* é chamada para adicionar os documentos correspondentes à palavra no vetor *"documentos"* (Linha 5).

O serviço *"efetua busca"* passa, então, para a obtenção das métricas para cada documento retornado pelo índice invertido. Isso é realizado através da busca dessas informações no banco de dados (Passo 2.3). Com as métricas obtidas, o próximo passo é gerar o valor do IDF (Seção 2). Esse processo é detalhado no Algoritmo 4. O processo de cálculo do IDF começa com a definição de um dicionário vazio, *"idfs"*, para armazenar os valores calculados (Linha 1). Em seguida, o algoritmo conta o número total de documentos da lista de documentos (Linha 2). Para cada palavra presente no vetor de palavras, o algoritmo utiliza a função *"obterNumDocsComPalavra"* para determinar o número de documentos em que a palavra aparece (Linha

Algoritmo 3 – Algoritmo de busca de documentos.

```
1: documentos ← []
2: vetorPalavras ← efetuaPréProcessamento(buscaFeitaPeloUsuário)
3: para cada palavra no vetorPalavras faça
4:     se palavra está no índiceInvertido então
5:         documentos.acrescentar(índiceInvertido[palavra])
6:     finaliza se
7: finaliza para
```

Fonte: Autoria própria..

3 e 4). O valor do IDF é, então, calculado como o logaritmo do quociente entre o número total de documentos e o número de documentos que contêm a palavra. O resultado deste cálculo é armazenado no dicionário *"idfs"* para a palavra corrente (Linha 5).

Algoritmo 4 – Algoritmo de geração do IDF.

```
1: idfs ← {}
2: totalDeDocumentos ← contaTodosDocumentos(documentos)
3: para cada palavra no vetorPalavras faça
4:     numDocsComPalavra ← obterNumDocsComPalavra(documentos, palavra)
5:     idfs[palavra] ← log ( totalDeDocumentos / numDocsComPalavra )
6: finaliza para
```

Fonte: Autoria própria.

Após o cálculo do IDF, é necessário classificar os documentos mais relevantes para o usuário utilizando o cálculo de classificação BM25 (Seção 4). Este processo é ilustrado no Algoritmo 5. O processo começa com a definição das constantes b e $k_1 (Linha 1 e 2). Logo aps, criado um dicionrio vazio, chamado$ *"classificações"*$, para armazenar os resultad$
$seo passo 2.4, onde sero recuperados os documentos a serem exibidos para o usurio.$

Algoritmo 5 – Algoritmo de classificação.

1: b ← 0.75
2: k_1 ← 1.2
3: classificações ← {}
4: **para** cada palavra em vetorPalavras **faça**
5: **para** cada documento em documentos **faça**
6: tamanhoDoDoc ← obtemTamanhoDoDocumento(documento)
7: tamanhoMédioDoc ← $\frac{tamanhoDoDocumento}{totalDeDocumentos}$
8: frequenciaPalavraDoc ← obterFrequenciaPalavraDoc(documento, palavra)
9: classificação ← $idfs[palavra] \cdot \left(\frac{frequenciaPalavraDoc \cdot (k_1 + 1)}{frequenciaPalavraDoc + k_1 \cdot (1 - b + b \cdot \frac{tamanhoDoDoc}{tamanhoMedioDoc})} \right)$
10: **se** doc está no classificações **então**
11: classificações[doc] = classificação + classificações[doc]
12: **senão,**
13: classificações[doc] = classificação
14: **finaliza se**
15: **finaliza para**
16: **finaliza para**
17: ordenaDoMaiorParaMenor(classificações)

Fonte: Autoria própria.

5 RESULTADOS

Esse capítulo apresenta a análise de desempenho da arquitetura SIRA e do ELS em relação à capacidade de recuperação de documentos e ao consumo de recursos. O objetivo não é alcançar resultados melhores que o ELS, mas mostrar que a SIRA cumpre seu papel em relação à capacidade de recuperação de documentos.

A arquitetura SIRA foi executada no ambiente de computação sem servidor da AWS, chamado Lambda, e o ELS foi instalado em um servidor EC2 na AWS com 2 CPUs virtuais, 2 GB de memória RAM e sistema operacional linux. Foi utilizado um computador MacBook Pro com um processador M1 e 8GB de memória RAM para atuar como cliente e obter os resultados da SIRA e do ELS. A conexão à Internet foi feita por meio de uma rede Wi-Fi de 5 GHz, utilizando uma conexão de fibra óptica de 500 Mbps.

5.1 Análise da Capacidade de Recuperação de Documentos

O campo da RI tem se beneficiado significativamente da alta disponibilidade de conjuntos de dados para realizar avaliações. No entanto, a variedade de formatos desses conjuntos de dados pode dificultar o uso em experimentos e avaliações. Para resolver esse problema, foi utilizado o pacote *ir datasets* para Python. O pacote fornece uma interface comum para avaliação de classificação de RI e conjuntos de dados de treinamento. O pacote se encarrega de baixar os dados (incluindo documentos, consultas, julgamentos de relevância, etc.) quando disponíveis em fontes públicas (MACAVANEY *et al.*, 2021).

Para avaliar o desempenho da SIRA e do ELS em relação à capacidade de recuperação de documentos, foram utilizadas as coleções de testes Cranfield e TREC-COVID. A coleção Cranfield é uma coleção de teste pioneira que permite medidas precisas para avaliação de sistemas de RI. A coleção contém 1398 documentos que são resumos de artigos científicos sobre aerodinâmica, 225 consultas e julgamentos de relevância (qrels) (MANNING, 2008). TREC-COVID é uma coleção de teste criada pela comunidade, onde reúne informações científicas relevantes sobre o COVID-19. A coleção contém 195 mil documentos, 50 consultas e julgamentos de relevância (qrels) (VOORHEES *et al.*, 2021).

Utilizar coleções de documentos extensas é crucial para avaliar a escalabilidade dos sistemas de RI. No entanto, obter julgamentos de relevância para todos os documentos presentes em uma coleção de grande porte pode ser inviável (TONON; DEMARTINI; CUDRÉ-MAUROUX, 2015). Neste trabalho, optou-se por utilizar uma coleção de teste menor, pois o objetivo é avaliar apenas a capacidade de recuperação de documentos. A análise da escalabilidade do sistema está fora do escopo desse trabalho.

5.1.1 Análise com a Coleção de Teste Cranfield

Foram realizadas 225 consultas da coleção de teste Cranfield nas duas arquiteturas, SIRA e ELS. Ambas arquiteturas utilizaram os mesmos valores para as constantes b (= 0,75) e $k_1 (k_1 = 1{,}2) no cl culo do BM25$.

O ELS foi capaz de responder a 10 consultas, o que equivale a 4,44% das buscas e a SIRA foi capaz de responder a 12 consultas, o que equivale a 5,33% das buscas. A Tabela 1 mostra o qid (identificador da busca) e qual arquitetura conseguiu encontrar documentos para cada qid.

Tabela 1 – Documentos recuperados por arquitetura na coleção de teste Cranfield

qid	SIRA	ELS
1	X	X
2	X	X
33	-	X
56	X	X
57	-	X
83	X	-
99	X	-
100	X	-
109	-	X
130	-	X
171	X	X
189	X	-
196	-	X
212	X	-
213	X	-
214	X	X
225	X	-

Fonte: Autoria própria.

As primeiras métricas avaliadas para ambas arquiteturas foram a precisão (*precision*), *recall* e F1. Essas avaliações são importantes pois conseguem responder qual a capacidade do sistema de retornar documentos relevantes para uma determinada busca. A Tabela 2 mostra os resultados para cada arquitetura avaliada. Na tabela são apresentados o identificador da consulta (qid), a quantidade de documentos relevantes para consulta (Relevantes), a quantidade de documentos recuperados por ambas as arquiteturas (Recuperados) e a quantidade de documentos relevantes que cada arquitetura conseguiu recuperar por consulta (Rel. Recuperados).

O Gráfico 1 apresenta a avaliação da métrica *recall*. É possível observar que o ELS obteve os melhores resultados de *recall* para os qid 1, 2, 33, 56, 57, 109, 130, 171, 196 e 214. Já a SIRA, obteve os melhores resultados para os qid 56, 83, 99, 100, 171, 189, 212, 214 e 225.

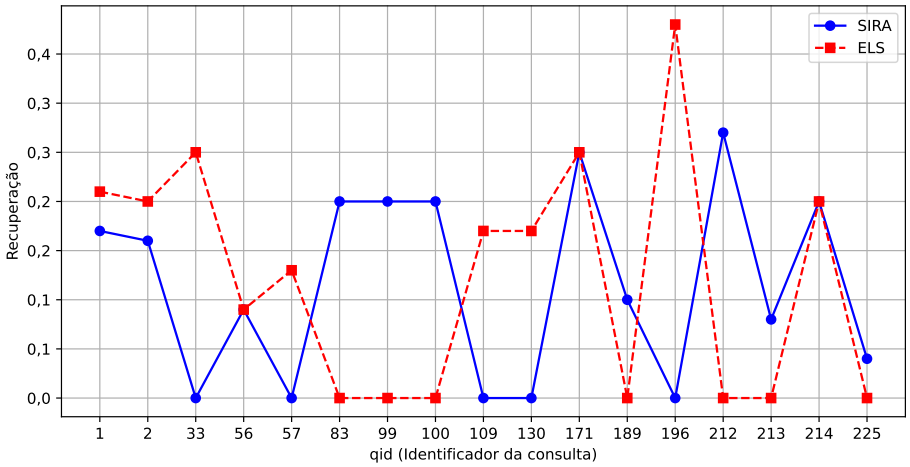
O Gráfico 2 demonstra a avaliação da métrica precisão. O ELS obteve os melhores resultados para os qid 1, 2, 33, 56, 57, 109, 130, 171, 196 e 214. A SIRA obteve os melhores resultados para os seguintes qid 56, 83, 99, 100, 171, 189, 212, 214 e 225.

Tabela 2 – Avaliação da recuperação, precisão e F1 para o ELS e SIRA na coleção de teste Cranfield

qid	Relevantes	Recuperados	Rel. Recuperados		Recuperação		Precisão		F1	
			ES	SIRA	ES	SIRA	ES	SIRA	ES	SIRA
1	29	10	6	5	0,21	0,17	0,60	0,50	0,31	0,26
2	25	10	5	4	0,20	0,16	0,50	0,40	0,29	0,23
33	4	10	1	0	0,25	0,0	0,10	0,0	0,14	0,0
56	11	10	1	1	0,09	0,09	0,10	0,10	0,10	0,10
57	15	10	2	0	0,13	0,0	0,20	0,0	0,16	0,0
83	5	10	0	1	0,0	0,20	0,0	0,10	0,0	0,13
99	5	10	0	1	0,0	0,20	0,0	0,10	0,0	0,13
100	10	10	0	2	0,0	0,20	0,0	0,20	0,0	0,20
109	6	10	1	0	0,17	0,0	0,10	0,0	0,12	0,0
130	6	10	1	0	0,17	0,0	0,10	0,0	0,12	0,0
171	4	10	1	1	0,25	0,25	0,10	0,10	0,14	0,14
189	10	10	0	1	0,0	0,10	0,0	0,10	0,0	0,10
196	13	10	5	0	0,38	0,0	0,50	0,0	0,43	0,0
212	15	10	0	4	0,0	0,27	0,0	0,40	0,0	0,32
213	12	10	0	1	0,0	0,08	0,0	0,10	0,0	0,09
214	5	10	1	1	0,20	0,20	0,10	0,10	0,13	0,13
225	25	10	0	1	0,0	0,04	0,0	0,10	0,0	0,06

Fonte: Autoria própria.

Gráfico 1 – Avaliação da métrica *recall* na coleção de teste Cranfield

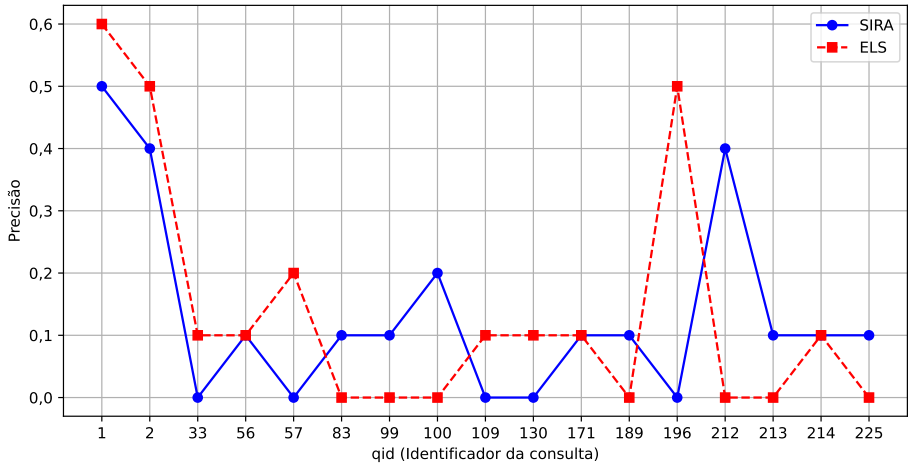


Fonte: Autoria própria.

O Gráfico 3 demonstra a avaliação da métrica F1. O ELS obteve os melhores resultados para os qid 1, 2, 33, 56, 57, 109, 130, 171, 196 e 214. A SIRA obteve os melhores resultados para os seguintes qid 56, 83, 99, 100, 171, 189, 212, 214 e 225.

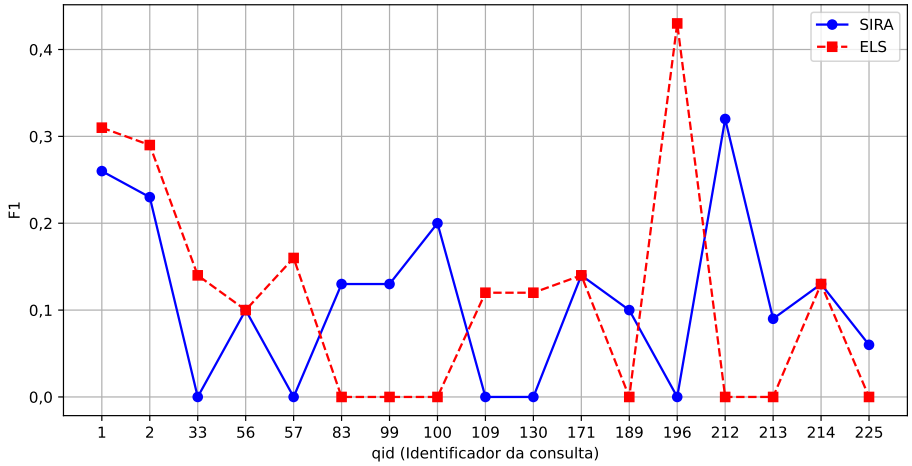
As avaliações de precisão e recuperação são importantes para verificar se os sistemas estão retornando documentos relevantes para a busca, mas elas não conseguem avaliar se esses documentos relevantes estão posicionados de forma adequada. Para avaliar esse as-

Gráfico 2 – Avaliação da métrica precisão na coleção de teste Cranfield



Fonte: Autoria própria.

Gráfico 3 – Avaliação da métrica F1 na coleção de teste Cranfield



Fonte: Autoria própria.

pecto, foram utilizadas as métricas de DCG e NDCG. A Tabela 3 mostra os resultados para cada arquitetura avaliada.

O Gráfico 4 demonstra a avaliação da métrica DCG. O ELS obteve os melhores resultados para os qid 1, 33, 56, 57, 109, 130, 171, 196. A SIRA obteve os melhores resultados para os qid 2, 83, 89, 100, 189, 212, 213, 214, 225.

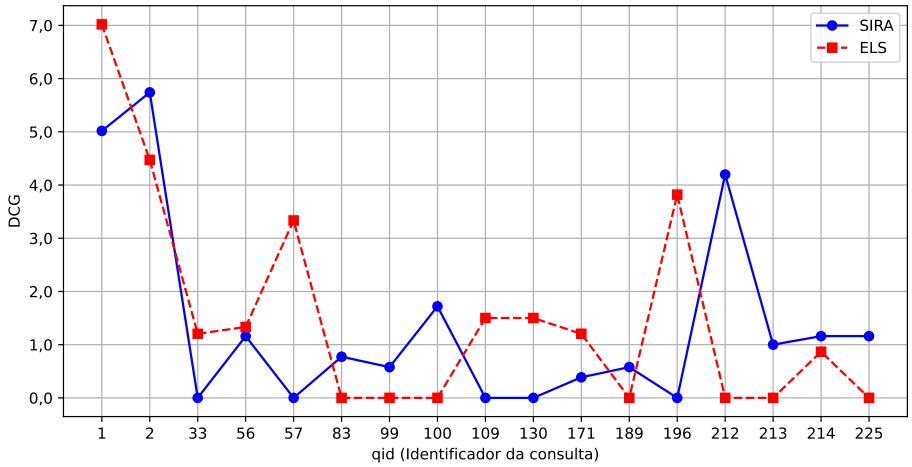
O Gráfico 5 demonstra a avaliação da métrica NDCG. O ELS obteve os melhores resultados para os qid 1, 33, 56, 57, 109, 130, 171, 196. á a SIRA, obteve os melhores resultados para os qid 2, 83, 89, 100, 189, 212, 213, 214, 225.

Tabela 3 – Avaliação das métricas DCG e NDCG para o ELS e SIRA na coleção de teste Cranfield

qid	IDCG	DCG		NDCG	
		ELS	SIRA	ELS	SIRA
1	24,75	7,02	5,02	0,28	0,20
2	20,04	4,47	5,74	0,22	0,29
33	14,76	1,20	0	0,19	0
56	4,74	1,33	1,16	0,09	0,08
57	5,67	3,33	0	0,18	0
83	4,74	0	0,77	0	0,16
99	5,67	0	0,58	0	0,10
100	10,01	0	1,72	0	0,17
109	10,01	1,50	0	0,29	0
130	3,20	1,50	0	0,20	0
171	10,68	1,20	0,39	0,38	0,12
189	10,68	0	0,58	0	0,05
196	15,40	3,82	0	0,30	0
212	15,40	0	4,20	0	0,27
213	10,95	0	1	0	0,09
214	10,95	0,87	1,16	0,12	0,16
225	22,63	0	1,16	0	0,05

Fonte: Autoria própria.

Gráfico 4 – Avaliação da métrica DCG na coleção de teste Cranfield

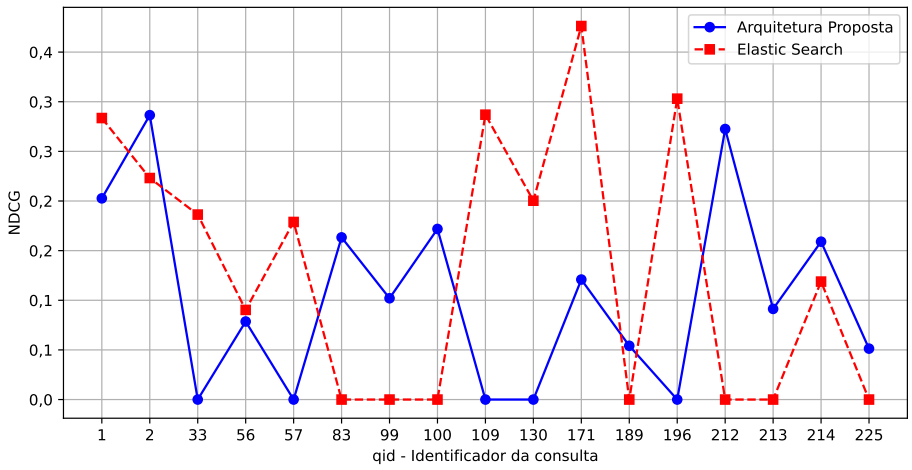


Fonte: Autoria própria.

5.1.2 Análise com a Coleção de Teste TREC-COVID

Foram realizadas 50 consultas da coleção de teste TREC-COVID nas duas arquiteturas, SIRA e ELS. Ambas arquiteturas utilizaram os mesmos valores para as constantes b (b = 0,75) e k_1 (k_1 = 1,2) no cálculo do BM25. O ELS e a SIRA foram capazes de responder as 50 consultas, o que equivale a 100% das buscas.

Gráfico 5 – Avaliação da métrica NDCG na coleção de teste Cranfield

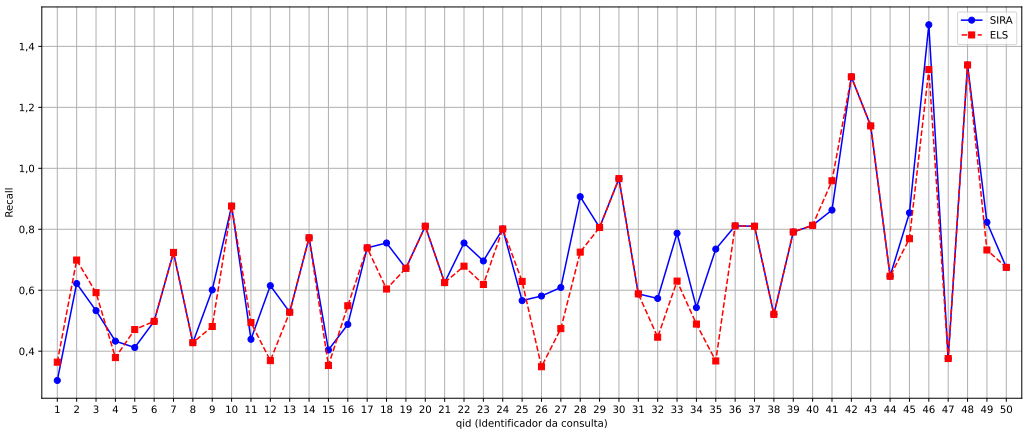


Fonte: Autoria própria.

A Tabela 4 mostra os resultados para cada arquitetura avaliada. Na tabela são apresentados o identificador da consulta (qid), a quantidade de documentos relevantes para consulta (Rel.), a quantidade de documentos recuperados por ambas as arquiteturas (Rec.) e a quantidade de documentos relevantes que cada arquitetura conseguiu recuperar por consulta (Rel. Recuperados).

O Gráfico 6 apresenta a avaliação da métrica *recall*. É possível observar que o ELS obteve os melhores resultados de *recall* para os qid 1, 2, 3, 5, 11, 16, 25 e 41. Já a SIRA, obteve os melhores resultados para os qid 4, 9, 12, 15, 18, 22, 23, 26, 27, 28, 32, 33, 34, 35, 45, 46 e 49.

Gráfico 6 – Avaliação da métrica *recall* na coleção de teste TREC-COVID



Fonte: Autoria própria.

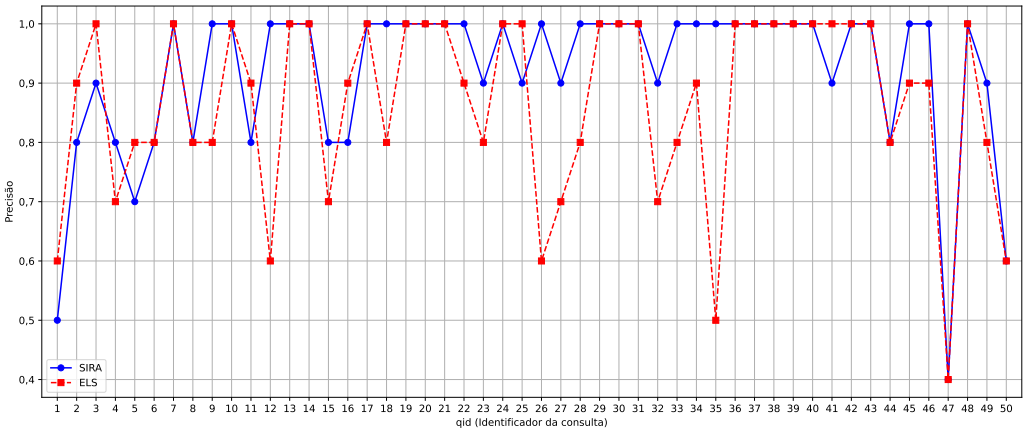
Tabela 4 – Avaliação da recuperação, precisão e F1 para o ELS e SIRA na coleção de teste TREC-COVID

qid	Rel.	Rec.	Rel. Recuperados		Recuperação		Precisão		F1	
			ES	SIRA	ES	SIRA	ES	SIRA	ES	SIRA
1	1647	10	6	5	0,00364	0,00304	0,6	0,5	0,0072	0,0060
2	1287	10	9	8	0,00699	0,00622	0,9	0,8	0,0139	0,0123
3	1688	10	10	9	0,00592	0,00533	1	0,9	0,0118	0,0106
4	1849	10	7	8	0,00379	0,00433	0,7	0,8	0,0075	0,0086
5	1697	10	8	7	0,00471	0,00412	0,8	0,7	0,0094	0,0082
6	1607	10	8	8	0,00498	0,00498	0,8	0,8	0,0099	0,0099
7	1382	10	10	10	0,00724	0,00724	1	1	0,0144	0,0144
8	1869	10	8	8	0,00428	0,00428	0,8	0,8	0,0085	0,0085
9	1664	10	8	10	0,00481	0,00601	0,8	1	0,0096	0,0119
10	1141	10	10	10	0,00876	0,00876	1	1	0,0174	0,0174
11	1821	10	9	8	0,00494	0,00439	0,9	0,8	0,0098	0,0087
12	1626	10	6	10	0,00369	0,00615	0,6	1	0,0073	0,0122
13	1893	10	10	10	0,00528	0,00528	1	1	0,0105	0,0105
14	1296	10	10	10	0,00772	0,00772	1	1	0,0153	0,0153
15	1981	10	7	8	0,00353	0,00404	0,7	0,8	0,0070	0,0080
16	1640	10	9	8	0,00549	0,00488	0,9	0,8	0,0109	0,0097
17	1353	10	10	10	0,00739	0,00739	1	1	0,0147	0,0147
18	1325	10	8	10	0,00604	0,00755	0,8	1	0,0120	0,0150
19	1489	10	10	10	0,00672	0,00672	1	1	0,0133	0,0133
20	1234	10	10	10	0,00810	0,00810	1	1	0,0161	0,0161
21	1600	10	10	10	0,00625	0,00625	1	1	0,0124	0,0124
22	1325	10	9	10	0,00679	0,00755	0,9	1	0,0135	0,0150
23	1293	10	8	9	0,00619	0,00696	0,8	0,9	0,0123	0,0138
24	1248	10	10	10	0,00801	0,00801	1	1	0,0159	0,0159
25	1590	10	10	9	0,00629	0,00566	1	0,9	0,0125	0,0113
26	1720	10	6	10	0,00349	0,00581	0,6	1	0,0069	0,0116
27	1477	10	7	9	0,00474	0,00609	0,7	0,9	0,0094	0,0121
28	1103	10	8	10	0,00725	0,00907	0,8	1	0,0144	0,0180
29	1241	10	10	10	0,00806	0,00806	1	1	0,0160	0,0160
30	1035	10	10	10	0,00966	0,00966	1	1	0,0191	0,0191
31	1701	10	10	10	0,00588	0,00588	1	1	0,0117	0,0117
32	1571	10	7	9	0,00446	0,00573	0,7	0,9	0,0089	0,0114
33	1270	10	8	10	0,00630	0,00787	0,8	1	0,0125	0,0156
34	1842	10	9	10	0,00489	0,00543	0,9	1	0,0097	0,0108
35	1360	10	5	10	0,00368	0,00735	0,5	1	0,0073	0,0146
36	1233	10	10	10	0,00811	0,00811	1	1	0,0161	0,0161
37	1234	10	10	10	0,00810	0,00810	1	1	0,0161	0,0161
38	1920	10	10	10	0,00521	0,00521	1	1	0,0104	0,0104
39	1264	10	10	10	0,00791	0,00791	1	1	0,0157	0,0157
40	1230	10	10	10	0,00813	0,00813	1	1	0,0161	0,0161
41	1043	10	10	9	0,00959	0,00863	1	0,9	0,0190	0,0171
42	769	10	10	10	0,01300	0,01300	1	1	0,0257	0,0257
43	878	10	10	10	0,01139	0,01139	1	1	0,0225	0,0225
44	1238	10	8	8	0,00646	0,00646	0,8	0,8	0,0128	0,0128
45	1171	10	9	10	0,00769	0,00854	0,9	1	0,0152	0,0169
46	680	10	9	10	0,01324	0,01471	0,9	1	0,0261	0,0290

Fonte: Autoria própria.

O Gráfico 7 demonstra a avaliação da precisão. O ELS obteve os melhores resultados para os qid 1, 2, 3, 5, 11, 16, 25 e 41. A SIRA obteve os melhores resultados para os qid 4, 9, 12, 15, 18, 22, 23, 26, 27, 28, 32, 33, 34, 35, 45, 46 e 49.

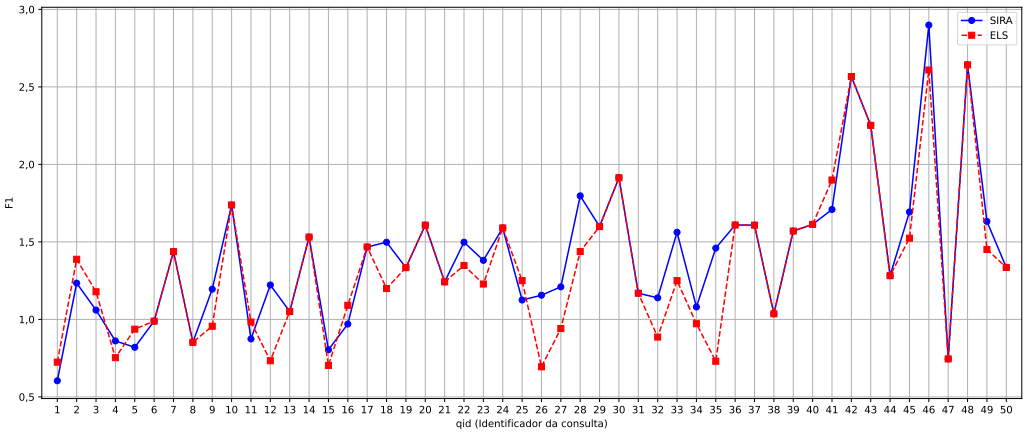
Gráfico 7 – Avaliação da métrica precisão na coleção de teste TREC-COVID



Fonte: Autoria própria.

O Gráfico 8 demonstra a avaliação da métrica F1. O ELS obteve os melhores resultados para os qid 1, 2, 3, 5, 11, 16, 25 e 41. A SIRA obteve os melhores resultados para os seguintes qid 4, 9, 12, 15, 18, 22, 23, 26, 27, 28, 32, 33, 34, 35, 45, 46 e 49.

Gráfico 8 – Avaliação da métrica F1 na coleção de teste TREC-COVID



Fonte: Autoria própria.

Os resultados de DCG e NDCG podem ser visualizados na Tabela 5 para cada arquitetura avaliada.

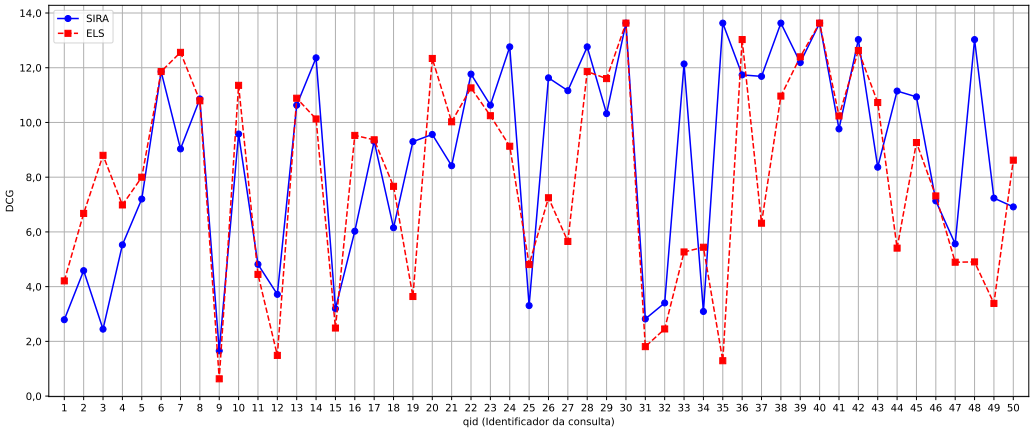
O Gráfico 9 demonstra a avaliação da métrica DCG. O ELS obteve os melhores resultados para os qid 1, 2, 3, 4, 5, 7, 10, 13, 16, 17, 18, 20, 21, 25, 29, 34, 36, 39, 41, 43, 46 e 50. A SIRA obteve os melhores resultados para os qid 8, 9, 11, 12, 14, 15, 19, 22, 23, 24, 26, 27, 28, 31, 32, 33, 35, 37, 38, 42, 44, 45, 47, 48 e 49.

Tabela 5 – Avaliação do DCG e NDCG para o ELS e SIRA na coleção de teste TREC-COVID

qid	IDCG	DCG		NDCG	
		ELS	SIRA	ELS	SIRA
1	13,6307	4,2103	2,7915	0,3089	0,2048
2	13,6307	6,6756	4,5835	0,4897	0,3363
3	13,6307	8,7994	2,4464	0,6456	0,1795
4	13,6307	6,9855	5,5272	0,5125	0,4055
5	13,6307	7,9990	7,2016	0,5868	0,5283
6	13,6307	11,8604	11,8604	0,8701	0,8701
7	13,6307	12,5621	9,0332	0,9216	0,6627
8	13,6307	10,7918	10,8604	0,7917	0,7968
9	13,6307	0,6309	1,6541	0,0463	0,1213
10	13,6307	11,3570	9,5783	0,8332	0,7027
11	13,6307	4,4477	4,8148	0,3263	0,3532
12	13,6307	1,4871	3,7183	0,1091	0,2728
13	13,6307	10,8859	10,6307	0,7986	0,7799
14	13,6307	10,1284	12,3620	0,7431	0,9069
15	13,6307	2,4871	3,1934	0,1825	0,2343
16	13,6307	9,5274	6,0258	0,6990	0,4421
17	13,6307	9,3662	9,3113	0,6871	0,6831
18	13,6307	7,6659	6,1522	0,5624	0,4514
19	13,6307	3,6389	9,2984	0,2670	0,6822
20	13,6307	12,3386	9,5621	0,9052	0,7015
21	13,6307	10,0237	8,4183	0,7354	0,6176
22	13,6307	11,2635	11,7668	0,8263	0,8633
23	13,6307	10,2473	10,6326	0,7518	0,7800
24	13,6307	9,1307	12,7635	0,6699	0,9364
25	13,6307	4,8065	3,3070	0,3526	0,2426
26	13,6307	7,2475	11,6307	0,5317	0,8533
27	13,6307	5,6521	11,1614	0,4147	0,8188
28	13,6307	11,8604	12,7635	0,8701	0,9364
29	13,6307	11,6029	10,3229	0,8512	0,7573
30	13,6307	13,6307	13,6307	1,0000	1,0000
31	13,6307	1,8068	2,8188	0,1326	0,2068
32	13,6307	2,4526	3,4031	0,1799	0,2497
33	13,6307	5,2669	12,1384	0,3864	0,8905
34	13,6307	5,4400	3,0925	0,3991	0,2269
35	13,6307	1,2920	13,6307	0,0948	1,0000
36	13,6307	13,0286	11,7379	0,9558	0,8611
37	13,6307	6,3185	11,6843	0,4636	0,8572
38	13,6307	10,9640	13,6307	0,8044	1,0000
39	13,6307	12,3977	12,1903	0,9095	0,8943
40	13,6307	13,6307	13,6307	1,0000	1,0000
41	13,6307	10,2379	9,7635	0,7511	0,7163
42	13,6307	12,6307	13,0286	0,9266	0,9558
43	13,6307	10,7279	8,3645	0,7870	0,6137
44	13,6307	5,4078	11,1480	0,3967	0,8179
45	13,6307	9,2645	10,9355	0,6797	0,8023
46	13,6307	7,3137	7,1328	0,5366	0,5233

Fonte: Autoria própria.

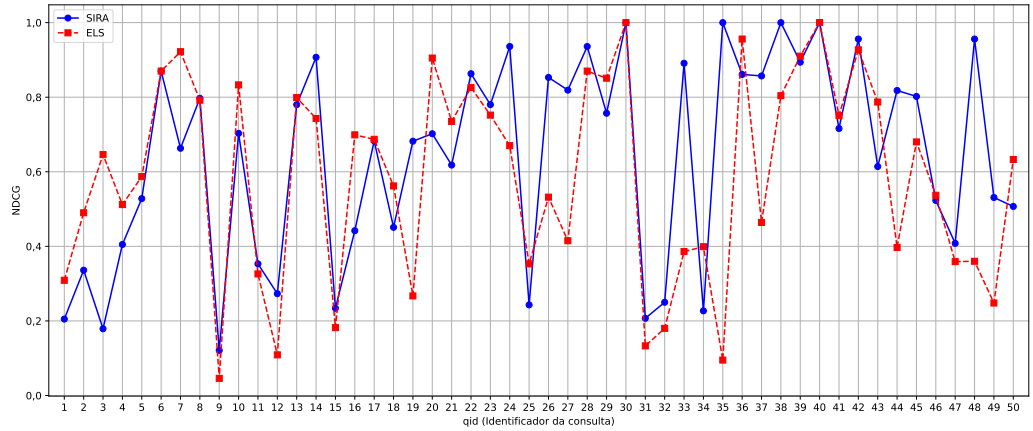
Gráfico 9 – Avaliação da métrica DCG na coleção de teste TREC-COVID



Fonte: Autoria própria.

O Gráfico 10 demonstra a avaliação da métrica NDCG. O ELS obteve os melhores resultados para os qid 1, 2, 3, 4, 5, 7, 10, 13, 16, 17, 18, 20, 21, 25, 29, 34, 36, 39, 41, 43, 46 e 50. A SIRA, obteve os melhores resultados para os qid 8, 9, 11, 12, 14, 15, 19, 22, 23, 24, 26, 27, 28, 31, 32, 33, 35, 37, 38, 42, 44, 45, 47, 48 e 49.

Gráfico 10 – Avaliação da métrica NDCG na coleção de teste TREC-COVID



Fonte: Autoria própria.

5.1.3 Discussão

Nas seções anteriores foram apresentados os resultados obtidos pela SIRA e ELS durante a recuperação de documentos ao utilizar duas coleções de testes distintas (Cranfield e TREC-COVID).

Os resultados demonstraram que a arquitetura SIRA obteve um desempenho comparável ao ELS, sendo capaz de responder a 5,33% das buscas para coleção de teste Cranfield,

enquanto o ELS respondeu a 4,44% das buscas. Para coleção de teste TREC-COVID, ambas arquiteturas conseguiram responder as 100% das buscas. Esses resultados indicam que a SIRA é eficaz em recuperar documentos relevantes, cumprindo seu papel como um sistema de recuperação da informação.

Pode-se perceber que a arquitetura SIRA e o ELS não obtiveram respostas relevantes em algumas pesquisas da coleção de teste Cranfield. Isso se deve ao fato de que essa coleção de teste foi elaborada por um ser humano, que possui uma compreensão semântica do texto. Assim, mesmo que o texto não contenha explicitamente as palavras-chave da busca, o ser humano consegue interpretar e classificar o documento como relevante com base apenas na semântica, ou seja, no significado do texto como um todo. Porém, o algoritmo BM25, utilizado neste estudo para classificar documentos, analisa a frequência de ocorrência das palavras-chave da pesquisa no documento, bem como a importância dessas palavras em relação a todos os documentos indexados (IDF - *Inverse Document Frequency*). A partir desses dados, ele calcula a relevância desse documento para a pesquisa.

Para ilustrar esse processo, vamos considerar a consulta com o id 4 da coleção de teste Cranfield, na qual a arquitetura SIRA e o ELS não conseguiram retornar resultado relevante. O motivo disso fica mais claro ao analisar o julgamento de relevância (Qrels) para a consulta de id 4 que, segundo o avaliador humano, esperava os seguintes ids de documentos: ’236’, ’166’, ’488’, sendo o documento ’236’ considerado o mais relevante para essa busca. Porém, o algoritmo BM25 retornou o documento de id 485 como mais relevante, conforme explicado em detalhes a seguir.

A consulta de id 4 continha o texto “*what problems of heat conduction in composite slabs have been solved so far.*”. Após o processamento do texto da busca, obteve-se os seguintes termos como resultado: [“*what*”, “*problem*”, “*heat*”, “*conduct*”, “*composit*”, “*slab*”, “*have*”, “*been*”, “*solv*”, “*so*”, “*far*”].

Depois de executar a busca na arquitetura SIRA para cada um desses termos processados, o BM25 retornou os seguintes dados, conforme apresentado na Tabela 6: o número de documentos em que a palavra (ou termo) aparece (N. Docs), o IDF, a frequência da palavra (TF) em cada documento e o valor de BM25 calculado para o termo em cada documento.

Vale ressaltar que a coleção de teste Cranfield apresenta um comprimento médio do documento (avgdl) de 103,03. Esta medida é um componente essencial para o cálculo do BM25 e é obtida pela soma de todas as palavras de todos os documentos indexados dividida pelo número total de documentos indexados.

Em termos do BM25, a relevância de um documento é determinada pela quantidade de palavras da busca presentes no documento e o alto IDF dessas palavras. Se olharmos para a palavra “*slab*”, por exemplo, vemos que ela tem um IDF alto. Em contraste, as palavras “*been*” e “*have*” possuem um IDF baixo. Isso se dá porque o cálculo do IDF penaliza palavras comuns que aparecem com frequência para que essas palavras não ofusquem palavras com poucas repetições em todos os documentos. Palavras menos frequentes geralmente carregam

Tabela 6 – Dados do BM25 por documento

Termos	N. Docs	IDF	Documento 236		Documento 485	
			TF	BM25	TF	BM25
what	16	4,41	1	4.47	0	0
problem	323	1,46	0	0	0	0
heat	306	1,51	0	0	2	2,65
conduct	158	2,17	0	0	1	3,15
composit	23	4,08	0	0	2	7,14
slab	14	4,57	0	0	2	7,98
have	371	1,32	0	0	0	0
been	383	1,29	0	0	0	0
solv	81	2,84	0	0	0	0
so	98	2,65	0	0	0	0
far	40	3,54	0	0	0	0

Fonte: Autoria própria.

um maior peso semântico em comparação a palavras que se repetem frequentemente. Portanto, um documento que contém um número maior de palavras da pesquisa e cujas palavras têm um IDF alto obterá uma classificação melhor.

O documento 485 possui um valor de BM25 significativamente mais alto em comparação ao documento 236, devido ao fato dele conter mais palavras-chave da busca e essas palavras apresentarem um IDF elevado. Por esse motivo, o documento 485 foi retornado pelo BM25 como o mais relevante. No entanto, o avaliador humano considera que, para a consulta com id 4, o documento 236 é o mais relevante. Isso se deve à análise semântica do texto feita pelo ser humano. Esse padrão é observado em todas as buscas realizadas que não retornaram documentos relevantes. É por essa razão que a arquitetura SIRA conseguiu fornecer respostas para 12 consultas, enquanto o ELS forneceu respostas para 10 consultas.

As métricas de precisão, *recall* e F1 foram utilizadas para avaliar a capacidade de ambos os sistemas em retornar documentos relevantes para uma consulta específica. Os resultados mostraram que o ELS obteve um desempenho melhor com a coleção de teste Cranfield, com um ganho de 3,23% na métrica F1, enquanto a SIRA obteve um desempenho melhor na coleção de teste TREC-COVID com um ganho de 5,42%. Isso indica que a arquitetura SIRA é capaz de fornecer resultados relevantes, mesmo quando comparada a uma ferramenta bem estabelecida como o ELS.

Outra métrica importante para avaliar a qualidade dos resultados de recuperação é o posicionamento dos documentos relevantes. Para isso, foram utilizadas as métricas DCG e NDCG. Ambas as arquiteturas apresentaram resultados variados para essas métricas. O ELS alcançou um resultado melhor na coleção de teste Cranfield, com um ganho de 28,22% para métrica NDCG, enquanto a SIRA obteve um melhor desempenho na coleção de teste TREC-COVID com ganho de 9,69%.

Existem diferenças nos resultados produzidos pelas arquiteturas SIRA e ELS, cujo o motivo está no processo de pré-processamento do texto. Esta etapa tem uma influência direta e significativa na frequência de repetições de uma palavra em um documento (TF) e no IDF dessa palavra. Para exemplificar essa diferença, considera-se a existência das palavras "*house*" (casa) e "*housing*" (abrigar). Embora sejam palavras distintas com diferentes significados, dependendo de como o pré-processamento do texto foi realizado, o resultado final pode ser "hous" para ambas as palavras. Isso implicaria em um TF de 2 no documento em vez de 1. Se esse padrão se repetir em todos os documentos para ambas as palavras, o número total de ocorrências dessas palavras aumenta, resultando em um IDF menor. Isso, por sua vez, diminui a importância dessas palavras entre todos os documentos e pode gerar resultados diferentes nas buscas entre as duas arquiteturas SIRA e ELS.

Outro fator que pode gerar diferenças nos resultados está relacionado com as características próprias do ELS. Este sistema possui diversos mecanismos para penalizar documentos que não correspondem perfeitamente à consulta. Por exemplo, ele pode penalizar documentos que contêm todos os termos da consulta, mas que não estão na ordem exata da consulta. Essas implementações fogem do escopo da arquitetura SIRA e podem contribuir para as diferenças observadas nos resultados.

Em conclusão, a arquitetura SIRA demonstrou ser uma solução promissora no campo da recuperação da informação, sendo capaz de fornecer resultados relevantes em várias consultas.

5.2 Análise do Consumo de Recursos

Nas seções anteriores pode-se constatar que a arquitetura SIRA provê um bom desempenho em relação à recuperação da informação ao compará-la com o ELS.

O objetivo da presente seção é analisar o desempenho das arquiteturas SIRA e ELS em relação ao consumo de recursos durante a recuperação da informação.

Utilizou-se a coleção de teste TREC-COVID, selecionando as 10 primeiras consultas da coleção, como apresentado na Tabela 7.

Iniciou-se a análise com a indexação de 5.000 documentos. Após concluir esses testes iniciais, prossegui-se com a indexação de blocos adicionais de 5.000 documentos cada, repetindo os mesmos testes até que um total de 100.000 documentos fossem indexados. Em cada indexação de 5.000 documentos, as arquiteturas SIRA e ELS retornaram os 10 documentos mais relevantes. Ao final do processo de indexação dos 100.000 documentos, a coleção de documentos consistia em um total de 11.416.784 palavras indexadas, cada uma com sua respectiva métrica de Frequência do Termo (TF). Ao sintetizar os procedimentos e resultados desta forma, buscou-se proporcionar uma representação clara e metódica do estudo, facilitando a sua compreensão e replicação.

As 10 consultas foram realizadas automaticamente através de um *script* em Python que também foi responsável pelo cálculo de três métricas de desempenho: tempo médio de proces-

Tabela 7 – Consultas da coleção de teste TREC-COVID realizadas durante a análise

Qid	Consulta
1	What is the origin of COVID-19?
2	How does the coronavirus respond to changes in the weather?
3	Will SARS-CoV2 infected people develop immunity? Is cross protection possible?
4	What causes death from Covid-19?
5	What drugs have been active against SARS-CoV or SARS-CoV-2 in animal studies?
6	What types of rapid testing for Covid-19 have been developed?
7	Are there serological tests that detect antibodies to coronavirus?
8	How has lack of testing availability led to underreporting of true incidence of Covid-19?
9	How has COVID-19 affected Canada
10	Has social distancing had an impact on slowing the spread of COVID-19?

Fonte: Voorhees *et al.* (2021).

samento das consultas em milissegundos (ms), a latência média da rede e a memória média consumida. A latência foi calculada subtraindo o tempo total de processamento do serviço de busca na AWS do tempo de resposta da requisição. O uso médio de memória do microsserviço de busca foi extraído dos *logs* fornecidos pela AWS após cada processamento. Todas essas métricas são apresentadas de forma detalhada na Tabela 8.

Tabela 8 – Análise de consumo de recursos das arquiteturas SIRA e ELS

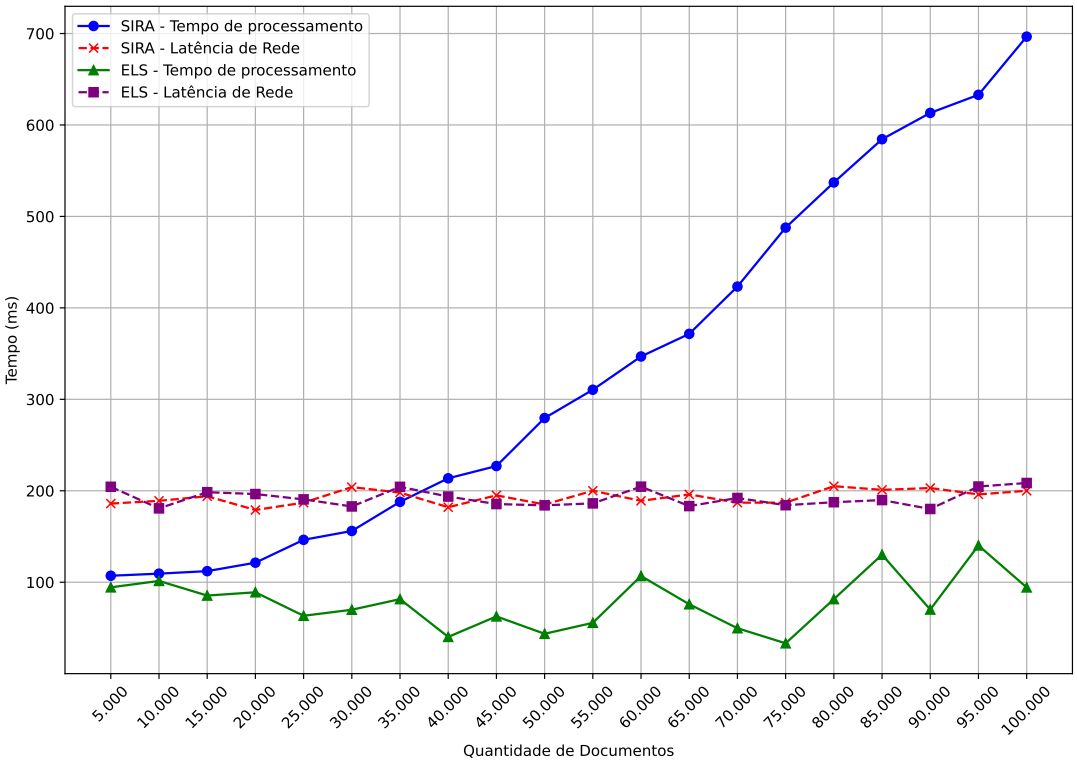
Qtd. Docs	SIRA			ELS		
	Processamento (ms)	Latência rede (ms)	Memória (MB)	Processamento (ms)	Latência rede (ms)	Memória (MB)
5.000	107,08	186	83	94,4	204	1.945,6
10.000	109,37	189	98	101,4	180	1.945,6
15.000	112,12	194	106	85,4	198	1.945,6
20.000	121,34	179	130	89	196	1.945,6
25.000	146,35	187	137	63,3	190	1.945,6
30.000	156,04	204	142	69,9	182	1.945,6
350.00	187,89	198	169	81,5	204	1.945,6
40.000	213,63	182	172	40,2	193	1.945,6
45.000	227,04	195	181	62,6	185	1.945,6
50.000	279,53	185	195	43,6	184	1.945,6
55.000	310,48	200	208	55,6	186	1.945,6
60.000	346,86	189	218	106,8	204	1.945,6
65.000	371,57	196	241	75,9	183	1.945,6
70.000	423,31	187	267	49,7	192	1.945,6
75.000	487,75	187	326	33,2	184	1.945,6
80.000	537,14	205	348	81,5	187	1.945,6
85.000	584,43	201	396	130,1	189	1.945,6
90.000	613,19	203	416	70,1	180	1.945,6
95.000	632,98	196	451	140,2	204	1.945,6
100.000	696,60	200	564	94,3	208	1.945,6

Fonte: Autoria própria.

O Gráfico 11 apresenta uma análise detalhada sobre como a quantidade de documentos a serem indexados afeta o desempenho do microsserviço de busca em relação ao tempo de processamento e latência de rede.

Na arquitetura SIRA, observa-se um incremento no tempo de processamento conforme a quantidade de documentos a serem indexados aumenta. Em média, há um aumento de 10,82% no tempo de processamento para cada bloco adicional de 5.000 documentos indexados. Este

Gráfico 11 – Tempo de processamento e latência de rede em relação à quantidade de documentos indexados.



Fonte: Autoria própria.

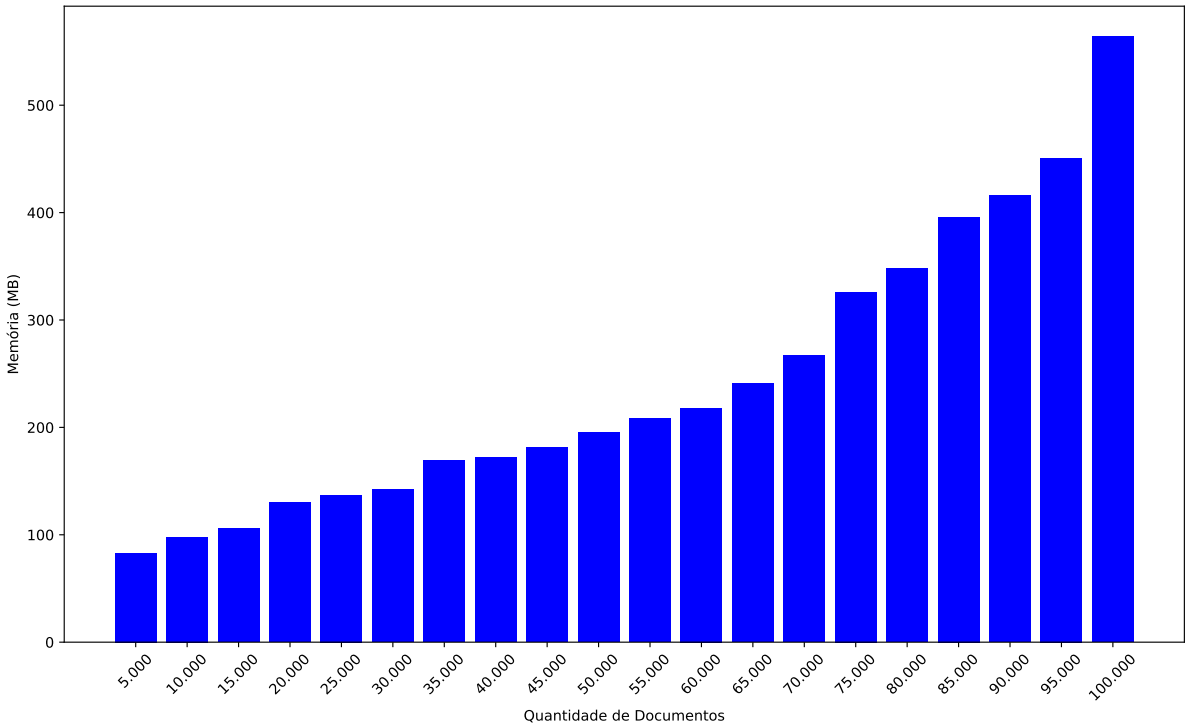
incremento pode ser atribuído à complexidade computacional associada ao cálculo do índice BM25, o qual precisa classificar um volume maior de documentos para determinar os 10 mais relevantes para a consulta do usuário. Em relação ao ELS, é possível observar que mesmo com um aumento na quantidade de documentos a serem indexados, ele pode obter uma redução no tempo de processamento. Isso pode ser visto, por exemplo, no tempo de processamento obtido ao indexar 60.000 e 75.000 documentos, onde o tempo de processamento foi de 106,8 ms e 33,2 ms, respectivamente. O uso de cache de consulta, responsável por armazenar resultados de buscas frequentes para reutilização em consultas futuras, contribuiu para essa eficiência do ELS. Em todos os testes realizados, o ELS proveu um menor tempo de processamento se comparado à SIRA.

Os resultados obtidos também demonstram que, em ambas as arquiteturas, a latência da rede não foi sensível ao tamanho do conjunto de documentos indexados. A média da latência de rede obtida na arquitetura SIRA foi de 195 ms e no ELS de 190 ms.

Em relação ao consumo de memória, o ELS aloca uma quantidade de memória desde o início de sua operação (2 GB), independentemente da quantidade de documentos a serem indexados. Essa característica se deve ao fato de o ELS ser construído sobre a linguagem Java, utilizando a Java Virtual Machine (JVM) para sua operação. A JVM fornece um ambiente de execução previsível e controlado. Ao alocar uma quantidade fixa de memória *heap* no início, a JVM pode garantir que haverá recursos suficientes para os objetos que a aplicação precisa criar,

evitando erros de falta de memória que poderiam ocorrer se a memória fosse alocada dinamicamente com base na demanda. Devido a essa peculiaridade, não é possível determinar com precisão a quantidade de memória consumida por cada consulta. Na arquitetura SIRA, como pode-se observar no Gráfico 12, conforme a quantidade de documentos a serem indexados aumenta, consome-se mais memória. A cada adição de 5.000 documentos ao índice, há um aumento médio de 10,51% no consumo de memória. Isso se deve a um aumento na quantidade de informações sobre métricas que devem ser mantidas na memória durante os cálculos do BM25.

Gráfico 12 – Consumo de memória em relação à quantidade de documentos indexados.



Fonte: Autoria própria.

Outra métrica importante para analisar o desempenho de uma arquitetura voltada para recuperação da informação é o tempo de resposta, calculado pela soma do tempo de processamento da busca e a latência de rede. Conforme estudos de Doherty e Sorenson (2015), a velocidade com que um sistema responde a uma busca é crucial para a experiência do usuário e ela é interpretada da seguinte forma: se o tempo de resposta é inferior a 300 ms, o usuário geralmente percebe a operação como sendo realizada de forma “instantânea”; Quando esse tempo varia entre 300 ms e 1 segundo, a operação é percebida como “imediata”. Acima de 1 segundo o usuário percebe que algum processamento está acontecendo.

Nos testes realizados com indexações de até 100.000 documentos, ambas as arquiteturas SIRA e ELS forneceram tempos de resposta abaixo de 1 segundo. O maior tempo de resposta obtido com o ELS foi de 344,20 ms ao indexar 95.000 documentos e com a SIRA foi de 896,60 ms ao indexar 100.000 documentos. No caso específico da SIRA, o tempo de resposta aumenta à medida que mais documentos precisam ser indexados devido ao tempo de proces-

samento, sendo capaz de fornecer uma operação “instantânea” ao usuário para indexações de até 10.000 documentos. Acima de 10.000 e até 100.000 documentos a serem indexados, a SIRA fornece uma operação “imediata”. Para indexar mais de 100.000 documentos, o tempo de resposta obtido pela SIRA é superior a 1 segundo. Uma estratégia para superar este desafio é a distribuição do processamento entre múltiplos microsserviços de busca para agilizar a entrega dos resultados. No entanto, a arquitetura, em sua concepção atual, não foi projetada para suportar tal distribuição, o que faz dessa uma área relevante para futuras pesquisas.

Um dos pilares fundamentais da arquitetura SIRA é a adoção do modelo de computação sem servidores. Esse modelo oferece duas vantagens significativas, um esquema de pagamento que é baseado em solicitações individuais e a capacidade de escalabilidade horizontal automática. Ao utilizar o serviço de computação sem servidor da AWS, os usuários beneficiam-se de até 1 milhão de requisições gratuitas por mês. Por outro lado, na computação provisionada, que envolve a alocação de recursos computacionais específicos, a execução do ELS em suas configurações mínimas requer um servidor EC2 com 2 CPUs virtuais e 2 GB de memória RAM, acarretando um custo mensal. Então, quanto mais próximo da ociosidade, melhor para a SIRA, pois sem requisições não haverá custo.

Através desta análise de consumo de recursos, fica evidente que o modelo de computação adotado pode ter impactos significativos tanto em termos de eficiência operacional quanto de custos. Assim, cada organização deve fazer uma avaliação criteriosa para determinar qual abordagem é mais adequada para suas necessidades.

6 CONCLUSÃO

A Recuperação de Informação (RI) é um campo de estudo que aborda o processamento, armazenamento, busca e classificação de informações relevantes. Sistemas de RI lidam com documentos não estruturados, escritos em linguagem natural (humana). Contudo, com o surgimento da computação sem servidor, iniciaram-se pesquisas sobre a aplicação desse modelo de computação na recuperação de informação, visando simplificar a gestão dos sistemas de RI.

Os estudos sobre RI utilizando computação sem servidor têm-se concentrado principalmente nas etapas de busca e classificação, o que motivou o desenvolvimento deste trabalho, que investiga métodos para processar, armazenar, buscar e classificar documentos empregando técnicas de RI. Como resultado, foi proposta uma arquitetura denominada SIRA (*Serverless Information Retrieval Architecture*), concebida como solução para a recuperação de informação utilizando a plataforma de computação sem servidor.

Para avaliar a arquitetura SIRA em relação à capacidade de recuperar documentos, foram utilizadas duas coleções de testes, Cranfield e TREC-COVID. As métricas utilizadas na avaliação foram: *Recall*, Precisão, F1, DCG e NDCG. Os resultados obtidos pela SIRA foram comparados com o *Elastic Search* (ELS), uma ferramenta padrão da indústria no campo da recuperação de informação que utiliza computação provisionada.

No que diz respeito à capacidade de recuperação de documentos pra coleção de teste Cranfield, o ELS respondeu a 4,44% das consultas, enquanto a SIRA respondeu a 5,33% das buscas. Para coleção de teste TREC-COVID, ambas as arquiteturas responderam a 100% das buscas. Em relação às métricas F1 e NCDG, o ELS obteve um desempenho melhor do que a SIRA na coleção de teste Cranfield, enquanto a SIRA apresentou um desempenho melhor do que o ELS na coleção TREC-COVID. Os resultados demonstraram um desempenho similar em ambas as arquiteturas em termos da recuperação de documentos relevantes.

O estudo também avaliou o desempenho da arquitetura SIRA em relação ao consumo de recursos físicos e financeiros e o comparou com o ELS.

Nas avaliações da arquitetura SIRA observou-se um aumento no tempo médio de processamento em cada bloco adicional de 5.000 documentos indexados. No ELS, foi possível observar que mesmo com um aumento na quantidade de documentos indexados, ele obteve uma redução no tempo de processamento devido ao seu mecanismo de cache. Em relação à latência de rede, esta não foi afetada pela quantidade de documentos indexados e ambas as arquiteturas SIRA e ELS obtiveram valores médios próximos de 200 ms. Outro recurso analisado foi a memória consumida. No ELS, a quantidade de memória é alocada no início de sua operação e se mantém fixa, independentemente da quantidade de documentos indexados. Na SIRA, a memória consumida aumenta à medida que mais documentos precisam ser indexados.

Sobre a experiência de usuários em relação ao tempo de resposta obtido em uma busca, os resultados indicam que as arquiteturas SIRA e ELS foram capazes de fornecer uma boa experiência ao usuário considerando uma indexação de até 100.000 documentos.

Por utilizar o modelo de computação sem servidor, a SIRA usufrui de um baixo custo. Isso se deve ao fato de a computação sem servidor utilizar um esquema de pagamento que se baseia em solicitações individuais. Sendo assim, se não houver solicitações, não haverá custo. Por outro lado, o ELS utiliza o modelo de computação provisionado que estipulará um pagamento fixo independente da quantidade consumida de recursos.

A arquitetura SIRA, embora ofereça muitos benefícios, apresenta algumas limitações quando aplicada em sistemas de recuperação de informações. Essas limitações acontecem pelas características específicas da computação sem servidor. Em ambientes de computação sem servidor, as funções podem ter tempos de inicialização variáveis, o que pode resultar em latência, um problema significativo para sistemas de recuperação de informações que dependem de respostas rápidas e consistentes. Além disso, plataformas de computação sem servidor geralmente têm limites estritos no tempo de execução das funções. Isso pode ser um desafio para tarefas de recuperação de informações que exigem processamento extenso, implicando em uma necessidade de fragmentação das tarefas ou resultando em interrupções.

As contribuições desta pesquisa ampliam o conhecimento sobre o uso da computação sem servidor na recuperação da informação e oferecem uma arquitetura alternativa que pode ser aplicada em diferentes contextos e necessidades. A SIRA tem potencial para impulsionar avanços significativos na área de recuperação da informação, facilitando a implementação de sistemas escaláveis.

A implementação completa e a avaliação da arquitetura proposta podem ser acessadas no GitHub, por meio do seguinte link: *https://github.com/sira-serverless-ir-arch*.

Como trabalhos futuros, pretende-se distribuir o processamento entre múltiplos microsserviços de busca para obter menores tempos de resposta e explorar a integração da arquitetura SIRA com tecnologias de inteligência artificial, como aprendizado de máquina e processamento de linguagem natural, para aprimorar a precisão e a relevância na classificação e recuperação de documentos. Também pretende-se avaliar a adaptabilidade e o desempenho da arquitetura SIRA em diferentes domínios de aplicação, como saúde, finanças e educação.

REFERÊNCIAS

ADITYA, P. *et al.* Will serverless computing revolutionize nfv? **Proceedings of the IEEE**, IEEE, v. 107, n. 4, p. 667–678, 2019.

AGACHE, A. *et al.* Firecracker: Lightweight virtualization for serverless applications. *In*: **17th {usenix} symposium on networked systems design and implementation ({nsdi} 20)**. [*S.l.*: *s.n.*], 2020. p. 419–434.

AL-MASRI, E. *et al.* A serverless iot architecture for smart waste management systems. *In*: IEEE. **2018 IEEE International Conference on Industrial Internet (ICII)**. [*S.l.*], 2018. p. 179–180.

ALEXANDER, S. **Microservices with Go**. [*S.l.*]: OREILLY, 2022.

ANAND, M. *et al.* Serverless bm25 search and bert reranking. *In*: **DESIRES**. [*S.l.*: *s.n.*], 2021. p. 3–9.

ANDI, H. K. Analysis of serverless computing techniques in cloud software framework. **Journal of IoT in Social, Mobile, Analytics, and Cloud**, v. 3, n. 3, p. 221–234, 2021.

AZMANDIAN, F. *et al.* Virtual machine monitor-based lightweight intrusion detection. **ACM SIGOPS Operating Systems Review**, ACM New York, NY, USA, v. 45, n. 2, p. 38–53, 2011.

BAEZA-YATES, R.; RIBEIRO-NETO, B. **Recuperação de Informação-: Conceitos e Tecnologia das Máquinas de Busca**. [*S.l.*]: Bookman Editora, 2013.

BALDINI, I. *et al.* Serverless computing: Current trends and open problems. *In*: **Research advances in cloud computing**. [*S.l.*]: Springer, 2017. p. 1–20.

BARESI, L.; MENDONÇA, D. F.; GARRIGA, M. Empowering low-latency applications through a serverless edge computing architecture. *In*: SPRINGER. **European Conference on Service-Oriented and Cloud Computing**. [*S.l.*], 2017. p. 196–210.

BASHIR, M. B. *et al.* Content-based information retrieval techniques based on grid computing: A review. **IETE Technical Review**, Taylor & Francis, v. 30, n. 3, p. 223–232, 2013.

BATES, M. **Understanding information retrieval systems**. [*S.l.*]: Auerbach Publications, 2011.

BIAŁECKI, A. *et al.* Apache lucene 4. *In*: **SIGIR 2012 workshop on open source information retrieval**. [*S.l.*: *s.n.*], 2012. p. 17.

BUTTCHER, S.; CLARKE, C. L.; CORMACK, G. V. **Information retrieval: Implementing and evaluating search engines**. [*S.l.*]: Mit Press, 2016.

BUYYA, R. *et al.* Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. **Future Generation computer systems**, Elsevier, v. 25, n. 6, p. 599–616, 2009.

CACHEDA, F.; PLACHOURAS, V.; OUNIS, I. A case study of distributed information retrieval architectures to index one terabyte of text. **Information processing & management**, Elsevier, v. 41, n. 5, p. 1141–1161, 2005.

CASANOVA, H. Distributed computing research issues in grid computing. **ACM SIGAct News**, ACM New York, NY, USA, v. 33, n. 3, p. 50–70, 2002.

CASTRO, P. *et al.* Serverless programming (function as a service). *In*: IEEE. **2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)**. [*S.l.*], 2017. p. 2658–2659.

CHOWDHURY, G. G. **Introduction to modern information retrieval**. [*S.l.*]: Facet publishing, 2010.

CONNELLY, S. **BM25 na prática — parte 2: o algoritmo BM25 e suas variáveis**. 2018. Disponível em: https://www.elastic.co/pt/blog/practical-bm25-part-2-the-bm25-algorithm-and-its-variables. Acesso em: 29/11/2022.

CRANE, M.; LIN, J. An exploration of serverless architectures for information retrieval. *In*: **Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval**. [*S.l.*: *s.n.*], 2017. p. 241–244.

DAHAN, U. Event-driven architecture: Soa through the looking glass. **Microsoft Architecture Journal**, v. 21, 2009.

DOHERTY, R. A.; SORENSON, P. Keeping users in the flow: mapping system responsiveness with user experience. **Procedia Manufacturing**, Elsevier, v. 3, p. 4384–4391, 2015.

DUNLOP, M. D. Time, relevance and interaction modelling for information retrieval. *In*: ACM NEW YORK, NY, USA. **ACM SIGIR Forum**. [*S.l.*], 1997. v. 31, n. SI, p. 206–213.

EDER, J.; KAPPEL, G.; SCHREFL, M. **Coupling and cohesion in object-oriented systems**. [*S.l.*], 1994.

EYK, E. V. *et al.* Serverless is more: From paas to present cloud computing. **IEEE Internet Computing**, IEEE, v. 22, n. 5, p. 8–17, 2018.

FREIRE, A. *et al.* Analysis of performance evaluation techniques for large-scale information retrieval. **INVITED SPEAKER: Analyzing the Performance of Top-K Retrieval Algorithms**, Citeseer, p. 2001, 2013.

GORMLEY, C.; TONG, Z. **Elasticsearch: the definitive guide: a distributed real-time search and analytics engine**. [*S.l.*]: "O'Reilly Media, Inc.", 2015.

GOYAL, A.; DADIZADEH, S. A survey on cloud computing. **University of British Columbia Technical Report for CS**, v. 508, p. 55–58, 2009.

HADDI, E.; LIU, X.; SHI, Y. The role of text pre-processing in sentiment analysis. **Procedia computer science**, Elsevier, v. 17, p. 26–32, 2013.

HARMAN, D. *et al.* Information retrieval: the early years. **Foundations and Trends® in Information Retrieval**, Now Publishers, Inc., v. 13, n. 5, p. 425–577, 2019.

HASSAN, H. B.; BARAKAT, S. A.; SARHAN, Q. I. Survey on serverless computing. **Journal of Cloud Computing**, SpringerOpen, v. 10, n. 1, p. 1–29, 2021.

JR, H. F. M. The use of the univ ac fac-tronic system in the library reference field. **American Documentation (pre-1986)**, Wiley Periodicals Inc., v. 4, n. 1, p. 16, 1953.

KADHIM, A. I. Term weighting for feature extraction on twitter: A comparison between bm25 and tf-idf. *In*: **2019 International Conference on Advanced Science and Engineering (ICOASE)**. [*S.l.*: *s.n.*], 2019. p. 124–128.

KRÓL, M.; PSARAS, I. Nfaas: named function as a service. *In*: **Proceedings of the 4th ACM Conference on Information-Centric Networking**. [*S.l.*: *s.n.*], 2017. p. 134–144.

LI, Z. *et al.* The serverless computing survey: A technical primer for design architecture. **ACM Computing Surveys (CSUR)**, ACM New York, NY, v. 54, n. 10s, p. 1–34, 2022.

LIDDY, E. D. Natural language processing. **In Encyclopedia of Library and Information Science, 2nd**, 2001.

LIN, J. A prototype of serverless lucene. **arXiv preprint arXiv:2002.01447**, 2020.

MACAVANEY, S. *et al.* Simplified data wrangling with ir$_{d}atasets$. *In* : **SIGIR**. $[S.l. : s.n.], 2021$.

MANNING, C. D. **Introduction to information retrieval**. [*S.l.*]: Syngress Publishing„ 2008.

MCGUIRK, K. Emanuel goldberg and his knowledge machine: Information, invention and political forces. **Online Information Review**, Emerald Group Publishing Limited, 2007.

MENKEN, I. **Cloud Computing-The Complete Cornerstone Guide to Cloud Computing Best Practices Concepts, Terms, and Techniques for Successfully Planning, Implementing... Enterprise IT Cloud Computing Technology**. [*S.l.*]: Emereo Pty Ltd, 2008.

MICHELSON, B. M. Event-driven architecture overview. **Patricia Seybold Group**, v. 2, n. 12, p. 10–1571, 2006.

O'HARA, J. Toward a commodity enterprise middleware: Can amqp enable a new era in messaging middleware? a look inside standards-based messaging with amqp. **Queue**, ACM New York, NY, USA, v. 5, n. 4, p. 48–55, 2007.

PATIL, M. *et al.* Inverted indexes for phrases and strings. *In*: **Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval**. [*S.l.*: *s.n.*], 2011. p. 555–564.

PENG, T.; LIU, L.; ZUO, W. Pu text classification enhanced by term frequency–inverse document frequency-improved weighting. **Concurrency and computation: practice and experience**, Wiley Online Library, v. 26, n. 3, p. 728–741, 2014.

ROBERTS, M.; CHAPIN, J. **What Is Serverless?** [*S.l.*]: O'Reilly Media, Incorporated, 2017.

ROBERTSON, S. Understanding inverse document frequency: on theoretical arguments for idf. **Journal of documentation**, Emerald Group Publishing Limited, 2004.

ROCHA, H. F. O.; ROCHA, H. F. O. Achieving resilience and event processing reliability in event-driven microservices. **Practical Event-Driven Microservices Architecture: Building Sustainable and Highly Scalable Event-Driven Microservices**, Springer, p. 275–321, 2022.

SALTON, G. **The SMART retrieval system—experiments in automatic document processing**. [*S.l.*]: Prentice-Hall, Inc., 1971.

SALTON, G.; MCGILL, M. J. **Introduction to modern information retrieval**. [*S.l.*]: mcgraw-hill, 1983.

SANDERSON, M.; CROFT, W. B. The history of information retrieval research. **Proceedings of the IEEE**, IEEE, v. 100, n. Special Centennial Issue, p. 1444–1451, 2012.

SANDERSON, M. *et al.* Test collection based evaluation of information retrieval systems. **Foundations and Trends® in Information Retrieval**, Now Publishers, Inc., v. 4, n. 4, p. 247–375, 2010.

SCHAATSBERGEN, B. **Behind the scenes, Lambda**. 2021. Disponível em: https://www.bschaatsbergen.com/behind-the-scenes-lambda/. Acesso em: 01/10/2022.

SEDEFOĞLU, Ö.; SÖZER, H. Cost minimization for deploying serverless functions. *In*: **Proceedings of the 36th Annual ACM Symposium on Applied Computing**. [*S.l.*: *s.n.*], 2021. p. 83–85.

SHI, Z.; KEUNG, J.; SONG, Q. An empirical study of bm25 and bm25f based feature location techniques. *In*: **Proceedings of the International Workshop on Innovative Software Development Methodologies and Practices**. [*S.l.*: *s.n.*], 2014. p. 106–114.

SINGH, V.; SAINI, B. An effective tokenization algorithm for information retrieval systems. **Departement of Computer Engineering, National Institute of Technology Kurukshetra, Haryana, India**, Citeseer, 2014.

SINGHAL, A. *et al.* Modern information retrieval: A brief overview. **IEEE Data Eng. Bull.**, v. 24, n. 4, p. 35–43, 2001.

SUO, K. *et al.* Tackling cold start of serverless applications by efficient and adaptive container runtime reusing. *In*: IEEE. **2021 IEEE International Conference on Cluster Computing (CLUSTER)**. [*S.l.*], 2021. p. 433–443.

THANAKI, J. **Python natural language processing**. [*S.l.*]: Packt Publishing Ltd, 2017.

THANGAVEL, D. *et al.* Performance evaluation of mqtt and coap via a common middleware. *In*: IEEE. **2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing (ISSNIP)**. [*S.l.*], 2014. p. 1–6.

TONON, A.; DEMARTINI, G.; CUDRÉ-MAUROUX, P. Pooling-based continuous evaluation of information retrieval systems. **Information Retrieval Journal**, Springer, v. 18, n. 5, p. 445–472, 2015.

TRAN, T. T. *et al.* An autonomous mobile robot system based on serverless computing and edge computing. *In*: IEEE. **2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)**. [*S.l.*], 2020. p. 334–337.

VAHIDINIA, P.; FARAHANI, B.; ALIEE, F. S. Cold start in serverless computing: Current trends and mitigation strategies. *In*: IEEE. **2020 International Conference on Omni-layer Intelligent Systems (COINS)**. [*S.l.*], 2020. p. 1–7.

VAQUERO, L. M. *et al.* **A break in the clouds: towards a cloud definition**. [*S.l.*]: ACM New York, NY, USA, 2008. 50–55 p.

VOORHEES, E. *et al.* Trec-covid: constructing a pandemic information retrieval test collection. *In*: ACM NEW YORK, NY, USA. **ACM SIGIR Forum**. [*S.l.*], 2021. v. 54, n. 1, p. 1–12.

VOORHEES, E. M. The philosophy of information retrieval evaluation. *In*: SPRINGER. **Workshop of the cross-language evaluation forum for european languages**. [*S.l.*], 2002. p. 355–370.

VOORHEES, E. M. The evolution of cranfield. *In*: **Information retrieval evaluation in a changing world**. [*S.l.*]: Springer, 2019. p. 45–69.

WANG, Y. *et al.* A theoretical analysis of ndcg type ranking measures. *In*: PMLR. **Conference on learning theory**. [*S.l.*], 2013. p. 25–54.

YANG, C.; HUANG, Q. **Spatial cloud computing: a practical approach**. [*S.l.*]: CRC Press, 2013.