SIRA: uma Arquitetura para Recuperação de Informação sem Servidor

Thiago Luiz Rodrigues
Programa de Pós-Graduação em Computação Aplicada (PPGCA).
Universidade Federal Tecnológica do Paraná - UTFPR
Curitiba, Brasil
thiago@rodriguesthiago.me

Ana Cristina Barreiras Kochem Vendramin
Programa de Pós-Graduação em Computação Aplicada (PPGCA).
Universidade Federal Tecnológica do Paraná - UTFPR
Curitiba, Brasil
criskochem@utfpr.edu.br

Luiz Nacamura Júnior
Programa de Pós-Graduação em Computação Aplicada (PPGCA).
Universidade Federal Tecnológica do Paraná - UTFPR
Curitiba, Brasil
nacamura@utfpr.edu.br

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 a serverless computing platform. 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 the TREC-COVID test collection. In addition to having the advantage of consuming resources only on demand, the SIRA architecture demonstrates more effectiveness in retrieving relevant documents, indicating its potential as a solution in the field of information retrieval.

PALAVRAS-CHAVE

Sistema Distribuído, Computação sem Servidor, Recuperação de Informação, Indexação de Documentos, Ranqueamento de Documentos

1 INTRODUÇÃO

A área de recuperação da informação tem suas raízes nas técnicas desenvolvidas pelos sumérios há 3.000 AC, que arquivavam e organizavam informações em tábuas de argila [1]. A partir de 1950, surgiram propostas de uso de computadores para a recuperação da informação [2]. Essa área de pesquisa trata do armazenamento, organização e recuperação de documentos de texto não estruturados em linguagem natural [3].

Sistemas de recuperação de informação são softwares que empregam as técnicas de recuperação da informação [4]. O sistema de recuperação da informação mais famoso entre o usuários da *web* é o *Google*. À medida que o volume de documentos em um sistema de recuperação de informação aumenta, milhares de computadores (recursos) são necessários para atender a carga de trabalho e entregar documentos de maneira mais rápida aos usuários [5]. Diante desse cenário, técnicas como a distribuição do processamento, comunicação assíncrona, escalabilidade horizontal e uso de *cache* se tornam essenciais [5].

O gerenciamento de uma infraestrutura capaz de suportar a recuperação da informação é uma tarefa complexa, envolvendo a gestão de servidores, balanceadores de carga e serviços de rede [6]. Alguns estudos, como os de [7] e [8], 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 da infraestrutura de tecnologia da informação para executar sua aplicação, envolvendo a escolha, configuração e gestão de servidores físicos e virtuais [9]. Porém, conforme aumenta-se a coleção de documentos de um sistema de recuperação da informação, provisionar novos recursos na nuvem deixa de ser uma tarefa trivial. Diante desse desafio, a computação sem servidor torna-se uma estratégia viável em relação ao uso da computação provisionada na recuperação da informação, pois não é preciso alocar uma infraestrutura para executar as aplicações [10].

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 [11]. A computação sem servidor usa a arquitetura orientada a eventos, pois as aplicações são executadas mediante um evento assíncrono ou síncrono [12].

Em 2017, iniciaram-se 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 [10]. Em [13], os autores adaptaram uma biblioteca para construção de sistemas de recuperação da informação inicialmente voltada para computação provisionada, para ser utilizada sobre a plataforma de computação sem servidor. Porém, nenhum trabalho até o momento abordou o tema de forma completa, envolvendo as etapas de indexação, busca e classificação de documentos.

Diante desse contexto, este artigo tem como objetivo propor uma arquitetura para recuperação da informação que funcione sobre a plataforma de computação sem servidor e que aborde as etapas de indexação, busca e classificação de documentos.

2 COMPUTAÇÃO SEM SERVIDOR

O modelo de computação sem servidor, também conhecido como "*serverless*", foi introduzido pela Amazon em 2014 e rapidamente adotado por outras grandes empresas de tecnologia, como Google e Microsoft [14]. A computação sem servidor é um modelo de computação em nuvem que permite a execução de *softwares* orientados a eventos sem a necessidade de provisionamento, configuração e gestão da infraestrutura de servidores [15].

O crescimento acelerado desse modelo resultou na disponibilidade de várias plataformas para computação sem servidor, oferecidas pelos principais provedores de serviços de nuvem. Os aplicativos desenvolvidos para o modelo de computação sem servidor, conhecidos como funções, são caracterizados por não manterem estado (*stateless*), ou seja, a memória é alocada para execução e liberada após a conclusão do processo [16]. Essas funções são ativadas por eventos, como mensagens publicadas em tópicos, arquivos adicionados a um serviço de repositório, ou solicitações via HTTP (*Hypertext Transfer Protocol*), proporcionando uma implantação rápida, com baixo custo e operação sem gerenciamento [16].

Um exemplo de arquitetura sem servidor é a *AWS (Amazon Web Services) Lambda* [17]. Quando a *AWS Lambda* foi construída, inicialmente foi escolhido o modelo virtualizado de servidor junto a contêineres *Linux*, onde 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 [17]. 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. Segundo [17], as chamadas ao sistema (*systemcalls*) poderiam ser limitadas, o que melhoraria a segurança ao executar funções não seguras, porém isso restringiria os recursos disponíveis às funções.

Pesquisadores da AWS procuraram uma solução de virtualização que pudesse gerar um isolamento forte na execução das funções, onde cada função poderia ter o seu próprio *kernel*, uma rápida inicialização e que executasse milhares de funções em um mesmo servidor sem desperdícios de recursos [17]. Diante dessa demanda, nasceu o projeto *Firecracker* [17].

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*) [17]. Um VMM é um *software* que permite a criação e gerenciamento de VMs em Linux, e gerencia a operação de um ambiente virtualizado em cima de uma máquina física [18]. O VMM gerencia a operação de *back-end* das VMs, alocando memória, armazenamento e outros recursos de entradas e saídas necessários [18]. Cada função é executada dentro de uma micro VM criada e gerenciada pelo *Firecracker*. Essas micro VMs possuem um micro *kernel*, contendo somente o necessário para executar funções. As micro VMs são compostas por *sandboxs*, que são ambientes de execução de funções. Cada ambiente é criado dentro de um *worker* que é uma instância EC2 não visualizada pelos usuários dos serviços *Lambdas Functions*. Cada *worker* suporta a execução de milhares de micro VMs.

As funções podem ser invocadas através de solicitações síncronas ou assíncronas [19]. Quando há uma solicitação síncrona de execução de uma função, a solicitação passa por um balanceador de carga que tem a função de equilibrar as chamadas entre diversos servidores. O balanceador de carga encaminha a solicitação para uma camada de *front-end* que verifica 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 um gerente de *workers* cuja função é verificar se existe um ambiente ativo de execução de funções. Se for a primeira execução da função, não vão existir *sandboxs* disponíveis. Então, o gerente de *workers* se comunica com o serviço de alocação responsável por iniciar um *sandbox* dentro de um *worker*. Após iniciado um *sandbox*, pode-se executar a função. Já nas execuções assíncronas das funções, o balanceador de carga encaminha a solicitação para o *front-end* externo, que a publica em um serviço de mensagens. Existe um componente chamado *poller* que recebe os eventos e os repassa para o *front-end* interno. A partir desse momento, o fluxo de comunicação seguirá conforme a chamada de invocação síncrona.

3 RECUPERAÇÃO DA INFORMAÇÃO

As técnicas de recuperação de informação foram projetadas para resolver os problemas de busca efetuada por um usuário em uma grande coleção de documentos de texto não estruturados escritos em linguagem natural (linguagem humana), tais como artigos, livros e páginas na web [4]. Um sistema de recuperação da informação precisa retornar documentos relevantes, mesmo que os termos que o usuário utiliza na consulta não estejam presentes nesses documentos [3].

Um sistema de recuperação da informação é composto pelos processos de coleta, indexação, busca e ranqueamento de documentos com base em uma pesquisa realizada pelo usuário [20].

O processo de coleta de documentos envolve a busca de documentos de várias fontes. Esses documentos são, em seguida, tratados e organizados em um repositório do sistema [20]. Uma vez organizados, a etapa de indexação começa com o pré-processamento dos documentos.

O PLN (Processamento de Linguagem Natural) é responsável por realizar o processamento de texto escrito em linguagem natural através das seguintes técnicas [21]:

- *Stopwords*: após os documentos estarem organizados em um repositório, inicia-se a remoção das palavras que não trazem significado ao texto, tais como artigos, conjunções e pronomes. Esse processo também é conhecido como *stopwords*;
- *Stemming*: após a remoção de algumas palavras, inicia-se o processo de remoção do radical das palavras, também conhecido como (*stemming*). O principal papel do *stemming* é remover vários sufixos para que haja mais redução no número de palavras [22];
- Extração de métricas: a extração de métricas (estatística) de um texto é utilizada para definir o quão relevante é um documento para uma determinada busca [23]. As seguintes métricas podem ser extraídas de um texto:
  - A primeira métrica extraída é a frequência do termo (TF – *Term Frequency*) em um determinado documento D $tf(t_i, D)$. Verifica-se quantas vezes uma determinada palavra se repete em um documento t_i, D e esse valor é dividido pelo total de palavras no documento $\sum t^i, D$, indicando quão importante é uma palavra para um documento específico [24];
  - A segunda métrica é a frequência inversa do documento (IDF - *Inverse Document Frequency*) $idf(t)$. Ela é calculada como o logaritmo do número de documentos na coleção de documentos N dividido pelo número de documentos onde o termo específico aparece df, indicando quão importante é uma palavra entre todos os documentos da coleção [25];

– A terceira métrica, chamada de frequência do termo inverso do documento (TF-IDF - *Term Frequency Inverse Document Frequency*), é calculada multiplicando o valor do TF pelo IDF. O TF-IDF é uma métrica estatística usada para descrever uma das maneiras pelas quais um mecanismo de pesquisa pode determinar se um texto é relevante em relação aos termos usados em uma consulta [26].

Como resultado dos processos de *stopwords*, *stemming* e extração de métricas, tem-se um índice global de palavras únicas, conhecido como índice invertido [20]. Para cada palavra única que ocorre em uma coleção de documentos, o índice invertido armazena uma lista dos documentos em que essa palavra ocorre [27]. As principais vantagens da utilização de um índice invertido são a possibilidade de manter todo o índice na memória RAM sem custos financeiros e computacionais elevados, pois o índice ocupa pouca memória e melhora o desempenho das buscas, dada a redução na dimensionalidade dos dados e pelo fato de o índice estar na memória [20]. Todo processo pode ser feito acessando somente o índice invertido, sem ser necessário acessar cada documento durante a busca [4].

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 processo de recuperação e ranqueamento de documentos é iniciado quando o usuário faz uma busca em linguagem natural. O modelo de busca de um por vez é a maneira mais simples de encontrar documentos dentro de um índice invertido [27]. O texto da busca é transformado em um vetor de palavras e o sistema consulta o índice invertido, comparando cada palavra do vetor da busca com as do índice invertido. Quando há correspondência, o sistema obtém os documentos que contêm as palavras da busca.

Após o retorno dos documentos pelo índice invertido, estes são submetidos a um algoritmo 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 com base na avaliação da similaridade de cada documento com a busca do usuário. Esse processo é chamado de ranqueamento [28].

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 1 [29].

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

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 [29]. 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 [30]. 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 o usuário em ordem decrescente de relevância, de acordo com os resultados do algoritmo de classificação [4].

3.1 Métricas de avaliação em sistemas de recuperação da informação

Avaliar um sistema de recuperação da informação é importante para determinar se ele está atendendo às necessidades dos usuários em relação às suas buscas. Existem 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 mede a qualidade da classificação de documentos. Como a avaliação do usuário é mais cara e difícil de ser feita corretamente, os pesquisadores costumam usar a avaliação do sistema [31].

Cleverdon (1966) identificou seis métricas para avaliar um sistema de recuperação da informação, incluindo *recall*, precisão (*precision*), cobertura, tempo de resposta, questões de apresentação e esforço do usuário. Porém, segundo [32], as principais métricas de avaliação de um sistema de recuperação da informação são *recall* e precisão, pois estas se concentram na eficácia da recuperação.

A métrica *recall* indica a capacidade de um sistema de recuperação da informação encontrar todos os documentos relevantes para uma determinada consulta [28]. O cálculo da métrica *recall* é realizado através da divisão do número de documentos relevantes que foram recuperados (R) pelo número total de documentos relevantes disponíveis para aquela busca específica na coleção de teste (C) [24].

A métrica de precisão indica a capacidade do sistema retornar apenas documentos relevantes para uma determinada consulta [24]. O cálculo da métrica de precisão é realizado através da divisão da quantidade de documentos relevantes que foram recuperados (R) pela quantidade total de documentos retornados pela busca (L) [24].

Com os resultados das métricas *recall* e precisão, é possível gerar a média harmônica, também conhecida como F1 (equação 2), que é útil quando ambas as métricas são importantes para a avaliação de um sistema de recuperação de informação [24].

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

A avaliação utilizando a precisão e *recall* é importante, pois consegue-se avaliar se o sistema é capaz de retornar documentos relevantes. Porém, não se 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 permite que cada documento recuperado tenha um grau de relevância, enquanto a maioria das métricas de classificação tradicionais permite apenas relevância binária, além de considerar uma função de penalidade sobre a classificação, enquanto muitas outras métricas pesam uniformemente todas as posições [33]. Para realizar uma avaliação utilizando o NDCG, é preciso calcular o ganho acumulativo descontado (DCG - *Discounted Cumulative Gain*) e dividi-lo pelo ganho acumulativo descontado ideal (IDCG - *Ideal Discounted Cumulative Gain*). O DCG é uma métrica que mede

a qualidade relativa de uma lista de resultados de uma busca. O cálculo do DCG é realizado através da 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 [24]. A ponderação é fornecida por uma função de penalidade, que é uma função logarítmica $\log_2(n)$. O IDCG utiliza o mesmo cálculo do DCG, mas considera a ordem ideal de classificação dos documentos obtida da coleção de teste e não o resultado da busca [24]. A coleção de teste fornece uma lista de documentos contendo em qual posição eles devem aparecer para uma determinada busca e qual é a relevância do documento naquela posição. Então, o IDCG determina o valor máximo de DCG possível em uma determinada consulta [24].

4 TRABALHOS RELACIONADOS

O *Elastic Search* (ELS) é um sistema de recuperação da informação distribuído que utiliza a biblioteca *Apache Lucene* de modo a armazenar, pesquisar e analisar grandes volumes de dados não estruturados em tempo real [34]. 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 [34]. O *Apache Lucene* é uma biblioteca de pesquisa moderna e de código aberto construída em Java, a qual foi projetada para entregar resultados relevantes com alto desempenho para sistemas de recuperação da informação [35]. O *Apache Lucene* fornece interfaces de programação para processamento de linguagem, criação de índices, busca e classificação.

Em [10] foi proposta a primeira aplicação de recuperação da informação sobre a plataforma de computação sem servidor. Para implementação do projeto, foi utilizado o serviço de computação em nuvem da AWS. O projeto proposto abordou a etapa busca e classificação. Apesar do sistema possuir um índice, toda etapa de indexação de documentos não utilizou a computação sem servidor. Esse trabalho conseguiu demostrar que é possível a implementação de uma solução de recuperação da informação sobre computação sem servidor.

No trabalho de [36], foi realizado um experimento que adaptou o *Apache Lucene* para ser utilizado com a computação sem servidor. Este projeto, porém, apresentou limitações, como assumir índices estáticos e ter uma suposição de que todo o índice caberia em uma única instância Lambda com recursos de memória limitados.

Em [13], um protótipo foi desenvolvido considerando a etapa de busca de um sistema de recuperação da informação sobre a computação sem servidor, utilizando dois métodos de classificação de documentos. A primeira etapa da recuperação foi realizada utilizando o *Apache Lucene* com o algoritmo de classificação probabilístico BM25 e depois os documentos foram reordenados utilizando o modelo *MonoBERT*, o qual utiliza a arquitetura de transformadores (*transformers*). Os índices invertidos são construídos localmente e logo após são armazenados em um repositório de arquivos na nuvem. Apesar dos avanços, esse protótipo também encontrou desafios, principalmente relacionados à latência e custos financeiros excessivos.

Até o momento, foram publicadas poucas pesquisas que propõe uma arquitetura para sistemas de recuperação de informações utilizando a computação sem servidor. Além disso, nenhuma das arquiteturas propostas aborda as técnicas de indexação, centralizando a atenção somente nas fases de busca e classificação utilizando a computação sem servidor.

A principal contribuição desse trabalho é dar continuidade às pesquisas realizadas envolvendo o uso da computação sem servidor na recuperação da informação e propor uma arquitetura para recuperação da informação, chamada SIRA (*Serverless Information Retrieval Architecture*), que aborde as etapas de indexação, busca e classificação.

5 ARQUITETURA SIRA

A arquitetura SIRA (*Serverless Information Retrieval Architecture*) incorpora a plataforma sem servidor AWS Lambda. Esta plataforma favorece a implementação de microsserviços, garantindo escalabilidade e flexibilidade na gestão dos serviços. A escolha deste modelo de computação também foi motivada pela sua capacidade de escalar horizontalmente os recursos sem a necessidade de estabelecer regras específicas de escalabilidade. Como resultado, a arquitetura é capaz de lidar com picos inesperados de demanda de recursos, evitando interrupções do serviço de recuperação da informação.

Todo acesso aos recursos passará 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 é utilizado o *DynamoDB*. O *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.

É utilizado o SNS (*Simple Notification Service*) como serviço de mensagens no modelo *publisher/subscriber* para mensagens do tipo eventos. O uso do SNS permite aplicar a técnica de *fanout* para distribuir uma mensagem para diversos consumidores, garantindo escalabilidade e desacoplamento entre os sistemas. Para mensagens do tipo comando, é utilizado o SQS (*Simple Queue Service*) que trabalha no modelo de filas. O modelo de filas do SQS auxilia na manutenção da ordem das mensagens e garante o processamento único de cada mensagem. O SNS não garante a retenção de longo prazo das mensagens. Em contraste, o SQS pode reter mensagens por períodos configuráveis, com um máximo de 14 dias. Ao utilizar o SNS em conjunto com o SQS, é possível alcançar maior resiliência, pois as mensagens que falham no processamento inicial podem ser reprocessadas a partir da fila SQS.

A arquitetura SIRA é dividida em duas etapas, conforme pode ser visto na Figura 1. A primeira etapa é responsável pela indexação dos documentos, tendo a finalidade de receber um documento, persistilo em um banco de dados, extrair métricas do texto e criar um índice invertido. A segunda etapa 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, classificar os melhores documentos e, por último, mostrar o resultado para o usuário.

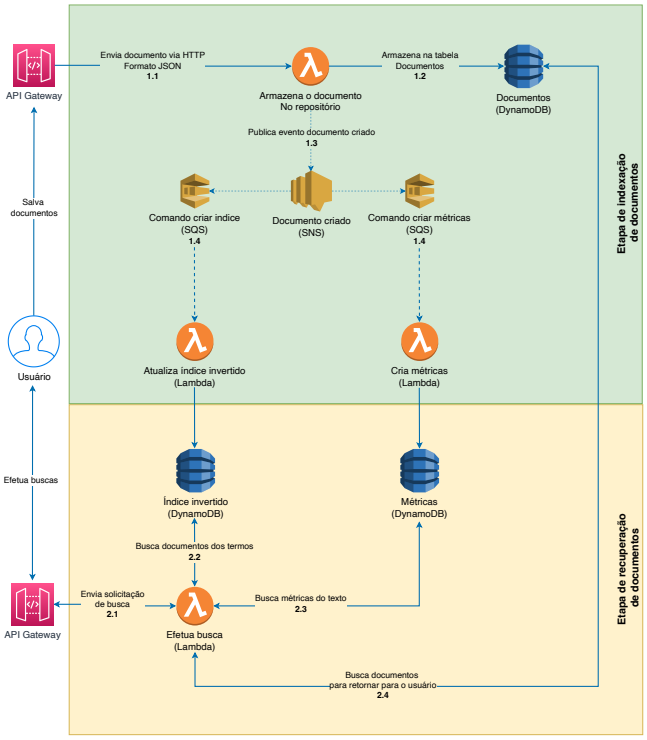

Figura 1: Arquitetura SIRA

A indexação de documentos na arquitetura SIRA ocorre através de uma série de etapas consecutivas. Primeiramente, um usuário submete documentos de texto para um *API Gateway* por meio de uma requisição HTTP do tipo POST. O *API Gateway*, no passo 1.1, gera um evento que executa o serviço de armazenamento de documentos, recebendo um documento por vez para otimização do processamento e controle de custos. Este documento, salvo em um banco de dados para recuperação posterior (passo 1.2), gera um novo evento (passo 1.3) que é distribuído pelo SNS simultaneamente para duas filas SQS. As mensagens dessas filas são consumidas pelos serviços "Atualiza índice invertido" e "Cria métricas" (passo 1.4). O serviço de atualização do índice invertido realiza um pré-processamento do documento, eliminando caracteres especiais e palavras de baixa qualidade semântica, e atualiza o índice invertido no banco de dados. Paralelamente, o serviço de criação de métricas também pré-processa o documento e gera métricas de frequência do termo no documento (TF), persistindo essas informações em um banco de dados de métricas para uso na etapa de recuperação de documentos.

No processo de recuperação de documentos, o usuário faz 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 em linguagem natural. No passo 2.1, um evento síncrono é acionado para executar o serviço denominado *"efetua busca"*. Durante o passo 2.2, esse serviço realiza o pré-processamento da busca. Esse procedimento inclui a eliminação de caracteres especiais e a geração de um vetor de palavras-chave. A seguir, o sistema itera pelas palavras-chave da consulta do usuário, verificando a presença de cada termo no índice invertido. Se uma palavra é localizada no índice invertido, os identificadores dos documentos correspondentes são adicionados a um vetor de documentos. No passo 2.3, o serviço consulta o banco de dados para obter as métricas de TF para cada documento retornado. Utilizando essas métricas, o serviço inicia o cálculo do IDF para as palavras-chave da consulta. Após o cálculo do IDF, os documentos são ordenados por sua relevância utilizando o algoritmo BM25. No passo 2.4, o serviço acessa o banco de dados de documentos para obter os textos completos dos documentos mais relevantes que são, então, exibidos ao usuário.

6 RESULTADOS

Essa seção 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. Para avaliação das arquiteturas foi utilizada a coleção de testes TREC-COVID. TREC-COVID é uma coleção de teste criada pela comunidade que mantém informações científicas relevantes sobre o COVID-19. A coleção contém 195 mil documentos, 50 consultas e julgamentos de relevância (qrels) [37].

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.

6.1 Análise da capacidade de recuperação de documentos

Para analisar a capacidade de recuperação de documentos da SIRA e do ELS foram realizadas 50 consultas da coleção de teste TREC-COVID. 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.

A Figura 2 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 qid 4, 9, 12, 15, 18, 22, 23, 26, 27, 28, 32, 33, 34, 35, 45, 46 e 49. A arquitetura SIRA demonstrou superioridade na métrica F1 em relação ao ELS, apresentando um ganho médio de desempenho de 5,42%. Isso sugere que a SIRA pode fornecer resultados relevantes, mesmo quando comparada a uma ferramenta bem estabelecida como o ELS.

A Figura 3 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. A arquitetura SIRA também demonstrou superioridade na métrica NDCG em relação ao ELS, apresentando um ganho médio de desempenho de 9,69%.

Existem diferenças nos resultados produzidos pelas arquiteturas SIRA e ELS, cujo o motivo está no processo de pré-processamento

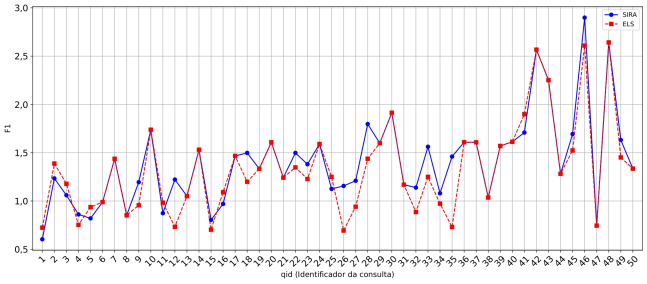

Figura 2: Avaliação da métrica F1 na coleção de teste TREC-COVID.

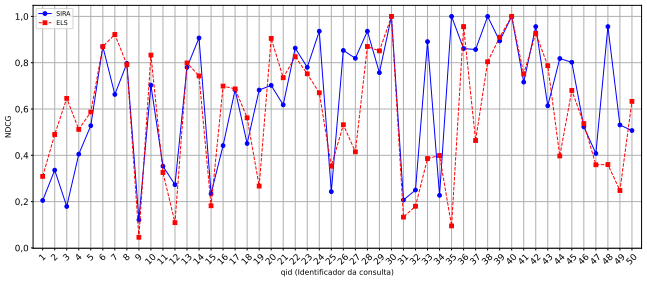

Figura 3: Avaliação da métrica NDCG na coleção de teste TREC-COVID.

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 ao invés 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.

6.2 Análise do Consumo de Recursos

O objetivo da presente seção é analisar o desempenho da SIRA 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. Essas 10 consultas foram realizadas automaticamente através de um *script* em Python que também foi responsável pelo cálculo de três métricas de desempenho: (i) tempo médio de processamento das consultas; (ii) latência média da rede; e (iii) memória média consumida.

Iniciou-se a análise com a indexação de 5.000 documentos. Após concluir esses testes iniciais, prosseguiu-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 TF.

A Figura 4 apresenta o desempenho de cada arquitetura em relação ao tempo de processamento e latência de rede. Na arquitetura SIRA, observa-se um aumento 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 aumento 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.

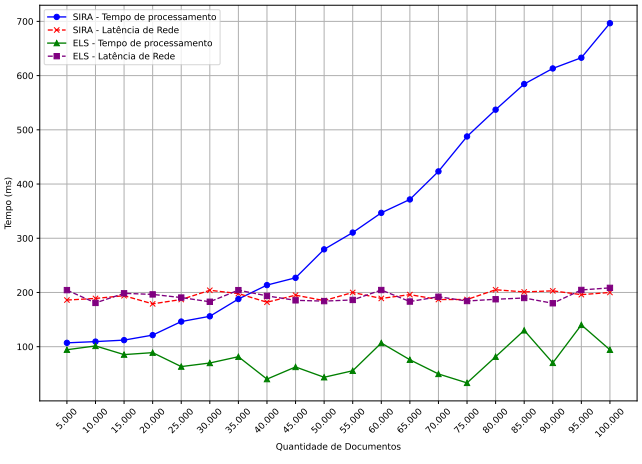

Figura 4: Tempo de processamento e latência de rede em relação à quantidade de documentos indexados.

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, 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 na Figura 5, 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.

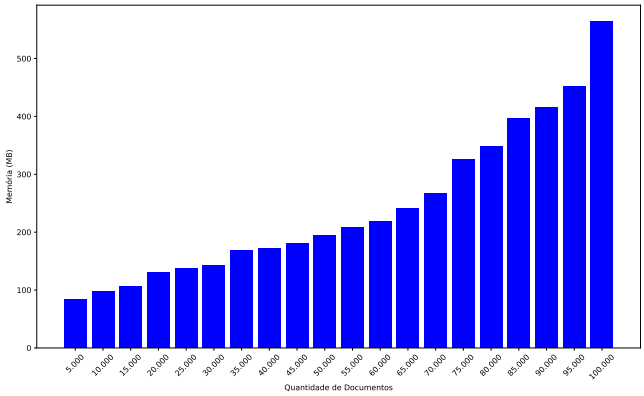

Figura 5: Consumo de memória em relação à quantidade de documentos indexados.

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 [38], 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 processamento, 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.

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.

7 CONCLUSÃO

Esse trabalho propôs uma arquitetura para recuperação da informação, chamada SIRA (*Serverless Information Retrieval Architecture*), que aborda as etapas de indexação, busca e classificação em uma plataforma de computação sem servidor. O desempenho da arquitetura SIRA foi comparado com o *Elastic Search* (ELS), uma ferramenta padrão da indústria no campo da recuperação de informação que utiliza computação provisionada, em relação à capacidade de recuperação de documentos e ao consumo de recursos. Para essa avaliação foi utilizada a coleção de teste TREC-COVID.

As métricas utilizadas na avaliação da capacidade de recuperação de documentos foram: F1 e NDCG. Ambas as arquiteturas responderam a 100% das buscas. Os resultados demonstraram um desempenho superior da arquitetura SIRA em relação ao ELS para ambas as métricas F1 e NDCG.

Em relação ao consumo de recursos, 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 contribuição desta pesquisa amplia o conhecimento sobre o uso da computação sem servidor na recuperação da informação e oferece 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.

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.

REFERÊNCIAS

[1] Mark Sanderson e W Bruce Croft. 2012. The history of information retrieval research. *Proceedings of the IEEE*, 100, Special Centennial Issue, 1444–1451.

[2] Donna Harman et al. 2019. Information retrieval: the early years. *Foundations and Trends® in Information Retrieval*, 13, 5, 425–577.

[3] Amit Singhal et al. 2001. Modern information retrieval: a brief overview. *IEEE Data Eng. Bull.*, 24, 4, 35–43.

[4] Christopher D Manning. 2008. *Introduction to information retrieval*. Syngress Publishing.

[5] Ana Freire, Fidel Cacheda, Vreixo Formoso e Víctor Carneiro. 2013. Analysis of performance evaluation techniques for large-scale information retrieval. *INVITED SPEAKER: Analyzing the Performance of Top-K Retrieval Algorithms*, 2001.

[6] Marcia Bates. 2011. *Understanding information retrieval systems*. Auerbach Publications.

[7] Fidel Cacheda, Vassilis Plachouras e Iadh Ounis. 2005. A case study of distributed information retrieval architectures to index one terabyte of text. *Information processing & management*, 41, 5, 1141–1161.

[8] Mohammed Bakri Bashir, Muhammad Shafie Abd Latiff, Aboamama Atahar Ahmed, Adil Yousif e Manhal Elfadil Eltayeeb. 2013. Content-based information retrieval techniques based on grid computing: a review. *IETE Technical Review*, 30, 3, 223–232.

[9] Özgür Sedefoğlu e Hasan Sözer. 2021. Cost minimization for deploying serverless functions. Em *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 83–85.

[10] Matt Crane e Jimmy Lin. 2017. An exploration of serverless architectures for information retrieval. Em *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, 241–244.

[11] Hari Krishnan Andi. 2021. Analysis of serverless computing techniques in cloud software framework. *Journal of IoT in Social, Mobile, Analytics, and Cloud*, 3, 3, 221–234.

[12] Zijun Li, Linsong Guo, Jiagan Cheng, Quan Chen, BingSheng He e Minyi Guo. 2022. The serverless computing survey: a technical primer for design architecture. *ACM Computing Surveys (CSUR)*, 54, 10s, 1–34.

[13] Mayank Anand, Jiarui Zhang, Shane Ding, Ji Xin e Jimmy Lin. 2021. Serverless bm25 search and bert reranking. Em *Desires*, 3–9.

[14] Hassan B Hassan, Saman A Barakat e Qusay I Sarhan. 2021. Survey on serverless computing. *Journal of Cloud Computing*, 10, 1, 1–29.

[15] Erwin Van Eyk, Lucian Toader, Sacheendra Talluri, Laurens Versluis, Alexandru Uta e Alexandru Iosup. 2018. Serverless is more: from paas to present cloud computing. *IEEE Internet Computing*, 22, 5, 8–17.

[16] Paul Castro, Vatche Ishakian, Vinod Muthusamy e Aleksander Slominski. 2017. Serverless programming (function as a service). Em *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2658–2659.

[17] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka e Diana-Maria Popa. 2020. Firecracker: lightweight virtualization for serverless applications. Em *17th {usenix} symposium on networked systems design and implementation ({nsdi} 20)*, 419–434.

[18] Fatemeh Azmandian, Micha Moffie, Malak Alshawabkeh, Jennifer Dy, Javed Aslam e David Kaeli. 2011. Virtual machine monitor-based lightweight intrusion detection. *ACM SIGOPS Operating Systems Review*, 45, 2, 38–53.

[19] Bruno Schaatsbergen. 2022. Behind the scenes, lambda.

[20] Ricardo Baeza-Yates e Berthier Ribeiro-Neto. 2013. *Recuperação de Informação: Conceitos e Tecnologia das Máquinas de Busca*. Bookman Editora.

[21] Elizabeth D Liddy. 2001. Natural language processing. *In Encyclopedia of Library and Information Science, 2nd.*

[22] Vikram Singh e Balwinder Saini. 2014. An effective tokenization algorithm for information retrieval systems. *Departement of Computer Engineering, National Institute of Technology Kurukshetra, Haryana, India.*

[23] Ammar Ismael Kadhim. 2019. Term weighting for feature extraction on twitter: a comparison between bm25 and tf-idf. Em *2019 International Conference on Advanced Science and Engineering (ICOASE)*, 124–128.

[24] Stefan Buttcher, Charles LA Clarke e Gordon V Cormack. 2016. *Information retrieval: Implementing and evaluating search engines*. Mit Press.

[25] Stephen Robertson. 2004. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation.*

[26] Tao Peng, Lu Liu e Wanli Zuo. 2014. Pu text classification enhanced by term frequency–inverse document frequency-improved weighting. *Concurrency and computation: practice and experience*, 26, 3, 728–741.

[27] Manish Patil, Sharma V Thankachan, Rahul Shah, Wing-Kai Hon, Jeffrey Scott Vitter e Sabrina Chandrasekaran. 2011. Inverted indexes for phrases and strings. Em *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, 555–564.

[28] Gobinda G Chowdhury. 2010. *Introduction to modern information retrieval*. Facet publishing.

[29] Zhendong Shi, Jacky Keung e Qinbao Song. 2014. An empirical study of bm25 and bm25f based feature location techniques. Em *Proceedings of the International Workshop on Innovative Software Development Methodologies and Practices*, 106–114.

[30] Shane Connelly. 2018. Bm25 na prática — parte 2: o algoritmo bm25 e suas variáveis. (abril de 2018). https://www.elastic.co/pt/blog/practical-bm25-part-2-the-bm25-algorithm-and-its-variables.

[31] Ellen M Voorhees. 2002. The philosophy of information retrieval evaluation. Em *Workshop of the cross-language evaluation forum for european languages*. Springer, 355–370.

[32] Mark D Dunlop. 1997. Time, relevance and interaction modelling for information retrieval. Em *ACM SIGIR Forum* número SI. Vol. 31. ACM New York, NY, USA, 206–213.

[33] Yining Wang, Liwei Wang, Yuanzhi Li, Di He e Tie-Yan Liu. 2013. A theoretical analysis of ndcg type ranking measures. Em *Conference on learning theory*. PMLR, 25–54.

[34] Clinton Gormley e Zachary Tong. 2015. *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. "O'Reilly Media, Inc.".

[35] Andrzej Białecki, Robert Muir, Grant Ingersoll e Lucid Imagination. 2012. Apache lucene 4. Em *SIGIR 2012 workshop on open source information retrieval*, 17.

[36] Jimmy Lin. 2020. A prototype of serverless lucene. *arXiv preprint arXiv:2002.01447*.

[37] Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff e Lucy Lu Wang. 2021. Trec-covid: constructing a pandemic information retrieval test collection. Em *ACM SIGIR Forum* número 1. Vol. 54. ACM New York, NY, USA, 1–12.

[38] Rina A Doherty e Paul Sorenson. 2015. Keeping users in the flow: mapping system responsiveness with user experience. *Procedia Manufacturing*, 3, 4384–4391.