feather: Máquina de busca de entidades baseada na Wikipedia e Wikidata

Matheus C. Candido[1]

[1]Instituto de Ciências Exatas e Informática – PUC Minas
Av Dom José Gaspar 500 – 30.535-901 – Belo Horizonte – MG – Brasil

matheus.candido@sga.pucminas.br

Resumo. *Esse trabalho apresenta o desenvolvimento e os resultados de uma máquina de busca de entidades realizado para a disciplina de Recuperação de Informação. O* dataset *utilizado é uma combinação dos* dumps *da Wikipedia e Wikidata.*

1. Introdução

Uma máquina de busca, *search engine*, motor de busca ou qualquer uma dessas variações de nome, é um programa de computador responsável por retornar documentos relevantes para consultas de palavras-chave. Esses motores surgiram logo após o surgimento da Internet (WWW), como ferramenta trivial na vida dos usuários da rede.

Com o crescimento da WWW, muitos desses buscadores foram perdendo utilidade já que não estavam preparados para lidar com tamanho volume de dados. Foi necessário desenvolver técnicas aprimoradas em diversas áreas da Ciência da Computação como Recuperação de Informação e Computação Linguística para geração de técnicas de ranqueamento e índices invertidos eficientes, bem como nas áreas de Sistemas Distribuídos e Armazenamento para controlar a performance desses motores.

Uma máquina de busca aberta é aquela que é capaz de indexar todas as páginas de um determinado domínio, sem filtro algum, por exemplo. Os documentos desse tipo de máquina são geralmente dados não estruturados e de marcação HTML, sendo necessário pré-processamento e inteligência de máquina para selecionar o que é relevante ou não no documento. Máquinas de busca de entidades funcionam de maneira diferente. Cada documento dessas máquinas, é na verdade (mas não limitado) a representação de um objeto (de tipos pré-definidos ou não), como pessoas, locais, empresas ou acontecimentos. Esses dados são armazenados de forma estrutural e muitas vezes servem de conteúdo para bases de conhecimento.

A Wikidata é uma base de conhecimento aberta e serve como armazenamento central de dados estruturados de projetos da Wikimedia, como Wikipedia, Wikivoyage, Wikisource e outros. Para evitar replicação de armazenamento, já que os projetos geram terabytes de conteúdo anualmente, são gerados *links* (ou ponteiros) para recursos nos projetos irmãos. Como os projetos mencionados em sua maioria são abertos (entradas podem ser editadas por qualquer pessoa), são oferecidos despejos de dados que podem ser gerados mensalmente ou semanalmente pela comunidade.

Nesse trabalho, exploramos a disponibilidade de tais despejos de dados (ou *dumps*), com os seguintes objetivos:

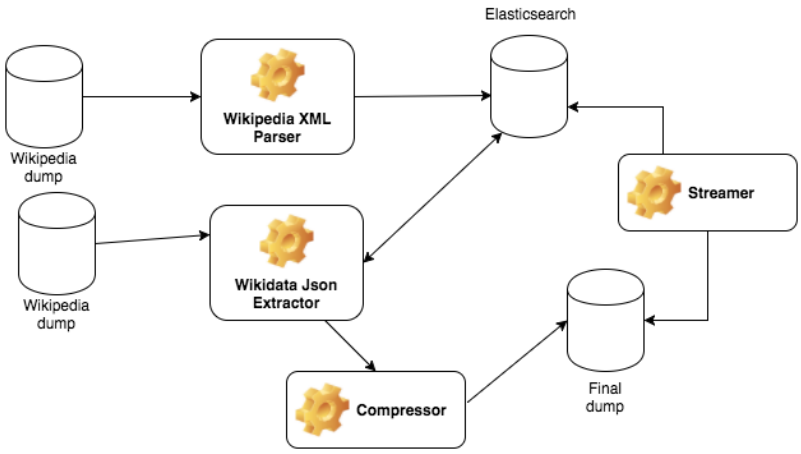

Figura 1. Arquitetura do coletor construído

- Construir uma máquina de busca de entidades com resultados satisfatórios
- Agrupar diversos tipos de informações e metadados disponíveis nos despejos estruturais da Wikidata com os artigos publicados na Wikipedia
- Implementar uma interface de interação com usuário simples e minimalista, que contenha métricas para cada consulta
- Facilitar a comunicação com a máquina de buscas utilizando uma arquitetura RESTful

2. Componentes

2.1. Coletor e Extrator

Coletores, *web crawlers, spiders* ou robôs são componentes indispensáveis em sistemas de buscas. O desenvolvimento dessas aplicações começou em 1993, quando o primeiro coletor foi implementado com o intuito de medir o tamanho da *World Wide Web* (WWW), descrito no relatório de [Gray 1993]. Vários trabalhos originaram-se dessa ideia e foram apresentados em conferências da WWW. Devido ao crescimento exponencial da Internet e o surgimento de novos motores de busca, os *crawlers* evoluíram até se tornarem escaláveis, rodando de forma distribuída e em várias máquinas. Muitos desses nem chegaram a ser descritos publicamente devido à natureza competitiva dos buscadores na época, com exceções como o do Google [Brin and Page 1998].

O coletor desse trabalho se diferencia um pouco dos *crawlers* originais. Ele usa *dumps* (registro de banco de dados) de dois projetos da organização Wikimedia para combinar e gerar documentos estruturados. A arquitetura é composta por quatro aplicações básicas, como indica a Figura 1.

2.1.1. Wikipedia XML Parser

O interpretador do *dump* XML da Wikipedia é uma modificação do `wikipedia-to-mongodb`, disponível no Github. As modificações incluem receber a entrada de dados do *stdin* e habilidade de ler o arquivo compactado. O programa original é um *parser* que lê arquivos XML na forma de *stream* e insere os documentos já estruturados no banco de dados MongoDB.

Propriedade Wikidata	Propriedade CombinedL	Descrição
P856	website	Website da entidade
P2013	facebook	Identificador de página ou perfil no Facebook
P2002	twitter	Identificador de perfil no Twitter
P2003	instagram	Identificador de perfil no Instagram
P2035	linkedin	Idenficador de página ou perfil no Linkedin
P2397	youtube	Idenficador de canal do Youtube
P345	imdb	Identificador do IMDB
P18	thumbnail	Link com imagem recuperada

Tabela 1. Mapeamento de propriedades do dump original e compinado

2.1.2. Wikidata JSON Parser e compreensão

O parser do *dump* da Wikidata funciona em conjunto com o Extrator do sistema de busca. Ela lê o arquivo JSON compactado (cada linha do arquivo é uma entrada), extrai as informações relevantes para o sistema (Tabela 1) e através do campo `sitelinks` recupera a entrada correspondente da Wikipedia, já indexada no Elasticsearch. A saída é um novo *dump* final, compactado, com os documentos a serem indexados novamente pelo Elasticsearch.

A saída não compactada do componente descrito na seção 2.1.2 é enviada para o `bzip2`, compressor de código aberto disponível em diversas plataformas.

2.1.3. Streamer

Este é um componente importantíssimo, atrelado ao indexador da máquina. Ele é responsável por enviar os documentos do novo arquivo de *dump* em forma de lotes para o Elasticsearch. A ferramenta utilizada foi o *stream2es*, um precursor do *Logstash*. Ele funciona de várias maneiras e uma delas é através de *streaming* de linhas vindas da `stdin`.

O que acontece nessa etapa é a descompactação dos documentos em tempo real através do comando `bzcat`, disponível no Compressor sendo passada para o *stream2es* pela `stdin`, que então é enviada para o Elasticsearch para indexação.

2.2. Indexador

O *software* responsável pela indexação dos documentos do sistema é o próprio Elasticsearch. Com apenas uma chamada na API do sistema é possível definir os campos que devem ou não ser processados e se é necessário manter *strings* em sua forma original (forma antes de serem *tokenizadas*).

2.3. Ranking

O Elasticsearch também responsável pelo ranqueamento dos documentos de uma pesquisa. Ele já vem com modelos que podem ser configurados através da API REST. O modelo usado para o sistema foi o Opaki BM-25 com parâmetros `k1 = 0` e `b = 0.75`.

2.4. Interface

A interface foi baseada no protótipo oferecido pelo professor, utilizando diversas bibliotecas de código aberto, como AngularJS, Bootstrap e ElasticUI. Para cada item de interface,

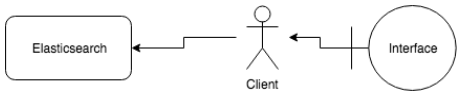

Figura 2. Arquitetura de Interface

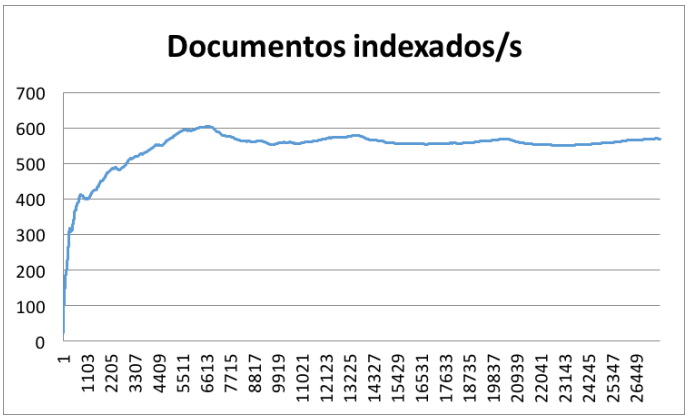

Figura 3. Quantidade de documentos indexados por segundo

existe um cliente do ElasticSearch para processar os resultados, como mostra a Figura 5.

Além de uma interface gráfica web com opções básicas, também foi produzido um aplicativo para a plataforma Android mais amigável e com mais recursos, tais como descritores de entidade e categoria extras.

3. Resultados

3.1. Coletor e Extrator

Nos testes realizados o sistema integrado do Coletor e Extrator geram um arquivo de 860 *megabytes* comprimidos, contendo entradas combinadas dos dois *dumps* descritos nas seções acima. A geração do arquivo final com os registros demora cerca de 10h em um processador Intel Core i5 de 2.4 GHz, com 8GB de memória RAM. O número total de registros é 21,354,787, com uma vazão de aproximadamente 593,2 documentos por segundo.

3.2. Indexador

O `stream2es`, responsável por indexar os documentos dentro do Elasticsearch, foi programado para executar operações em paralelo e configurado para executar em quatro *threads*. Após 121 minutos, todos os 21,354,787 documentos foram devidamente indexados, ocupando 10 *gibabytes* de armazenamento (documentos + índice invertido). Isso resulta em uma vazão de aproximadamente 2936.8 documentos por segundo.

3.3. Ranking

O componente de ranking também é outra adição do Elasticsearch. É possível configurar o fator de similaridade de campos específicos dos documentos, bem como ajustar o valor padrão de todos os campos.

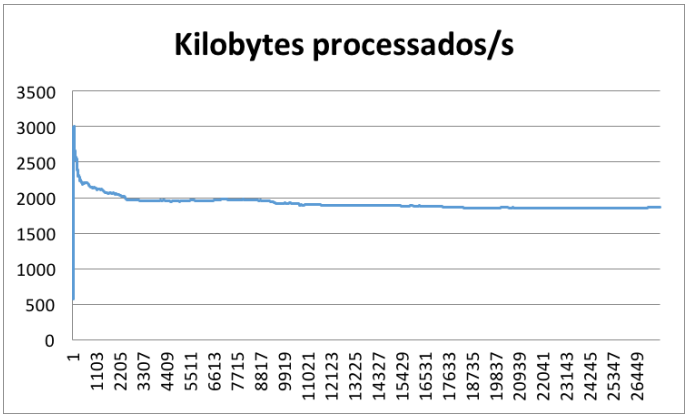

Figura 4. Quantidade de kilobytes processados por segundo

Tabela 2. Tabela com pesquisas teste de ranking

Query	# Docs relevantes	# Docs retornados	# Docs relevantes retornados	Precisão	revocação
Barack Obama	8	37	6	0.22	0.75
Obama	11	33	9	0.33	0.81
Dilma Roussef	3	10	3	0.3	1
Brin	1	3	1	0.33	1

Como mencionado na Seção 2.3, o modelo de ranking utilizado foi o BM25, com parâmetros `k1 = 0` e `b = 0.75`. Acima (Tabela 2) estão algumas *queries* e seus devidos valores de precisão e revocação.

3.4. Interface

A interface foi implementada seguindo o padrão do protótipo passado, com a exceção de alguns itens, que não puderam ser adicionados e com adição de novos.

Primeiramente a pesquisa é realizada nos campos `labels` e `aliases`, podendo ser alterada facilmente para cobrir "todos os campos". Essa escolha de implementação foi a que gerou melhores resultados. Os resultados aparecem instantaneamente na tela assim que usuário digita.

3.4.1. Itens extras

- Mostra imagem quando disponível
- Lista com links sociais (website, twitter, etc)
- É possível mostrar traduções breves dos títulos e descrições

A máquina pode ser acessada através do seguinte endereço: http://www.cassiano.me/feather.

4. Conclusão

Construir um buscador que seja capaz de responder pesquisas com qualidade e em tempo mínimo é uma tarefa árdua e que demanda bastante tempo e conhecimento. Esse trabalho

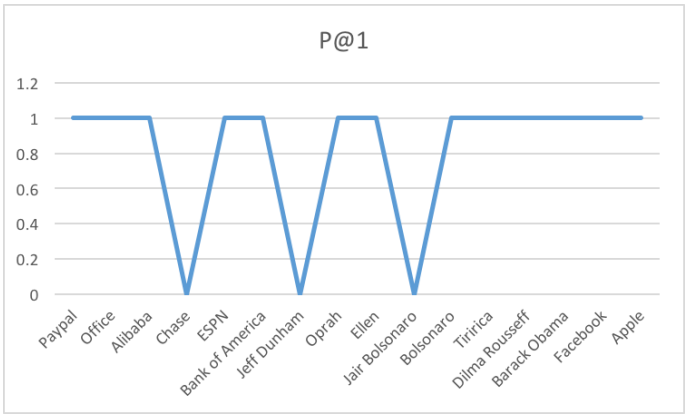

Figura 5. P@1 de consultas do site Alexa.com e do autor

foi uma tentativa de implementar um buscador de entidades seguindo diversas restrições impostas pelo professor.

Em geral, o sistema retorna documentos relevantes em boas posições de ranking. Nos testes realizados, muitas consultas tinham o documento mais relevante (o esperado) em primeiro lugar, mesmo o tamanho da coleção sendo absurdamente grande.

5. Trabalhos futuros

Melhorias no sistema de ranqueamento no sentido de adicionar modelos que trabalham com Aprendizado de Máquina podem apresentar melhores resultados, principalmente quando usando *Log* de consultas e recursos externos como os documentos mais vistos ou com mais referências na Wikipedia.

Referências

[Brin and Page 1998] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. In *Seventh International World-Wide Web Conference (WWW 1998)*.

[Gray 1993] Gray, M. (1993). Measuring the growth of the web. Disponível em: `http://www.mit.edu/people/mkgray/growth`. Acesso em: 06 mar. 2016.