Indexação com Filtros de Bloom

Natasha Rocha

Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil

Resumo Neste artigo irei implementar um índice baseado em filtros de Bloom, inspirado no BitFunnel, e outro baseado em listas invertidas para poder comparar os resultados. Falo sobre como funciona a indexação por assinatura, assim como as otimizações realizadas pel BitFunnel para resolver os problemas enfrentados por mim durante a implementação.

Keywords: Sistemas de Informação · Indexação de Documentos · Filtro de Bloom · Estruturas probabilísticas

1 Introdução

Lista invertida é a estrutura de dados mais comumente usada em sistemas comerciais de recuperação de informação. Uma outra possibilidade para a indexação seria a representação de documentos por assinaturas, mas pesquisas anteriores demonstraram que listas invertidas superam assinaturas em praticamente todos os critérios. [2]

Os desafios em implementar um sistema baseado em assinaturas, como, por exemplo, a natureza zipfiana da distribuição de frequência dos termos, ampla variedade no tamanho dos documentos, presença de faltos positivos, leitura excessiva de memória e difícil configuração, desencorajaram pesquisas na área e, consequentemente, sua adoção comercial.

Listas invertidas, no entanto possuem algumas desvantagens claras para buscas na web, como custos altos de manutenção e overhead de armazenamento. Tanto a inserção, como alteração ou deleção de documentos do índice possuem um custo muito elevado, que pode acabar se tornando proibitivo quando a coleção é composta por páginas da web, que está em constante expansão e alteração.

Problemas com custos operacionais motivaram a criação do BitFunnel pela Microsoft, que, após sua implementação, aumentou a capacidade de processamento dos servidores do Bing em até 10 vezes e diminuiu o tempo de processamento de consultas pela metade.

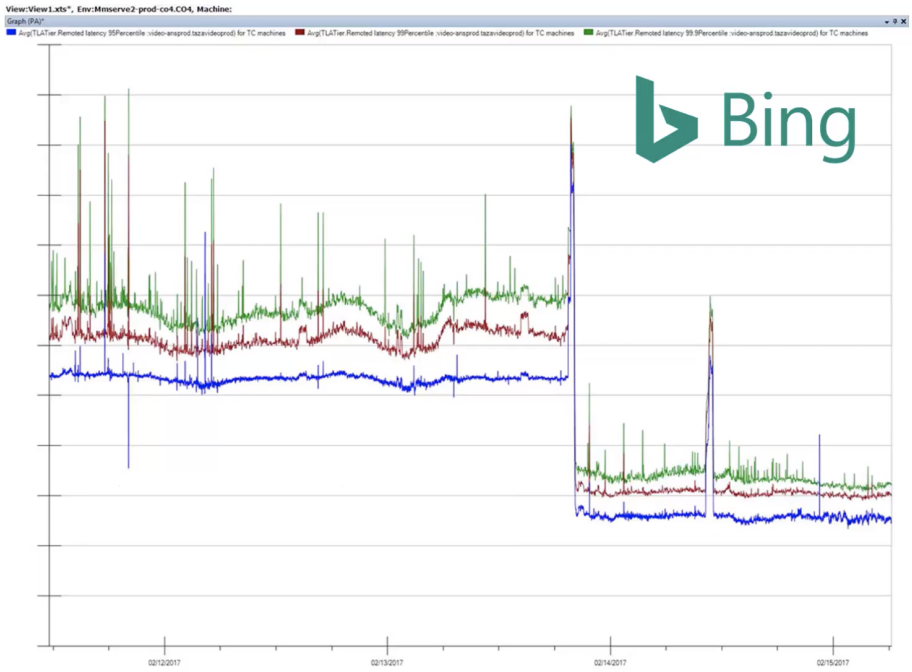

Figura 1. Tempo de processamento de consultas no Bing antes e após a substituição de um sistema baseado em listas invertidas pelo BitFunnel.[3]

Nesse artigo irei implementar minha própria versão de um sistema de informação com indexação por assinatura baseada em filtros de Bloom, inspirado no artigo sobre o BitFunnel. [1] O sistema tratará exclusivamente de buscas booleanas, desconsiderando a relevância de documentos.

2 Assinaturas Bit-String

Tabelas hash implementam alguma função de dispersão para definir a posição de um elemento em uma lista encadeada. Filtros de Bloom funcionam de forma similar.

Nele temos um vetor de bits, inicialmente setados em zero, para armazenar a assinatura dos nossos elementos. Ao contrário das tabelas hash, que usam apenas uma função de dispersão para cada termo, nos filtros de Bloom usamos mais de uma função para mapear o termo para algumas posições diferentes do vetor, que representarão aquele elemento - como uma assinatura.

Considere que temos um documento composto por apenas três termos: "big brown dog". Nós podemos representar esse documento por um vetor de bits com tamanho fixo: que será a assinatura do documento, ou filtro de Bloom. Para esse

exemplo digamos que o nosso vetor possui apenas 16 bits. Usando três funções de hash podemos representá-lo da seguinte forma:

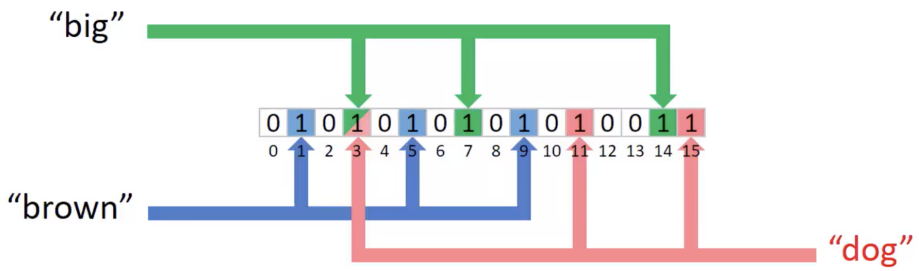

Figura 2. Filtro de Bloom com 16 bits, representando o documento ”big brown dog”.[3]

Note a existência de uma colisão na posição 3 do vetor. Quanto mais termos forem colocados no vetor maior é a possibilidade de encontrarmos colisões, o que aumenta a chance de encontrarmos falsos positivos. Um filtro totalmente cheio tem todos os seus bits setados em 1 e acusa possível match para qualquer entrada.

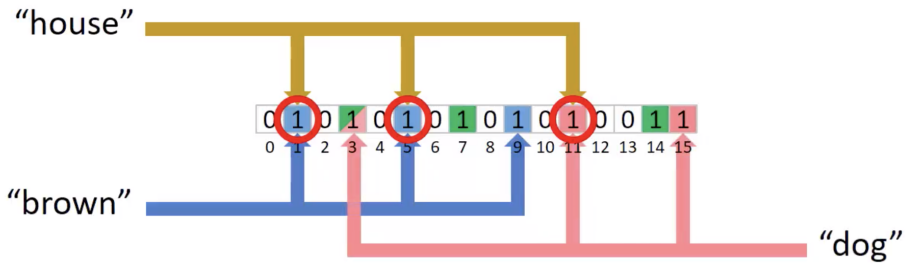

Figura 3. Bits dos termos ”brown”e ”dog”contribuíram para que nosso filtro considerasse ”house”como um possível membro do conjunto. [3]

Filtros de Bloom são estruturas probabilísticas que permitem a rápida verificação dos elementos presentes em um conjunto, além de serem estruturas razoavelmente compactas.

3 Implementação e Experimentação

Além das bibliotecas padrões de Python usei a biblioteca bitarray [4] para criar os vetores de bits do filtro de Bloom. Todo o código foi escrito em python. Foram criadas classes para representar os documentos (Documents), o índice baseado em assinaturas (SignatureIndex), o índice baseado em lista invertida (InvertedIndex) e os filtros de Bloom (BloomFilter). Nenhum dos índices foi otimizado.

3.1 Pré-processamento dos Dados e Consulta

Os documentos serão representados por strings. O único pré-processamento feito nas consultas foi a tokenização das palavras nos documentos. As consultas foram tokenizadas da mesma forma.

3.2 BloomFilter

Usei as funções de hash da biblioteca nativa de Python *hashlib*. A quantidade de funções usadas e o tamanho do filtro dependem da probabilidade esperada de falsos positivos e da quantidade de termos esperada. Esses foram os valores usados nos testes, que foram definidos arbitrariamente:

- *Probabilidade de falsos positivos:* 0.1
- *Quantidade de termos esperada:* 11160

3.3 Datasets

Usei dois datasets diferentes para comparar o desempenho dos sistemas: um da Amazon [5], com avaliações de produtos feitas por clientes, e outro com documentos mais longos, com artigos da Wikipedia [6].

Tabela 1. Informações sobre os datasets usados para testar os sistemas.

dataset	**descrição**	**nº docs**	**maior doc.**
reviews_Musical_Instruments_5	avaliações de instrumentos musicais da Amazon	10261	2090 termos
wiki-articles-1000	pequena coleção de artigos da Wikipedia, em inglês	10000	6113 termos

3.4 Indexação dos Documentos

A indexação dos documentos diretamente a partir dos arquivos json com os datasets consumia um tempo considerável, então optei por serializar os índices com o módulo *pickle*, nativo de Python. Houve uma redução considerável no tempo de carregamento inicial.

```
# Resultados para Dataset Wikipedia
Index loaded with pickle? False
Index processing time: 22.54014s

Index loaded with pickle? True
Index processing time: 1.461247s

# Resultados para Dataset Amazon
Index loaded with pickle? False
Index processing time: 8.536542s

Index loaded with pickle? True
Index processing time: 0.891091s
```

3.5 REPL

Para cada um dos datasets criei uma demo REPL (*read-eval-print-loop*). Para testar o código basta instalar a única dependência do projeto e executar um dos arquivos com as demos.

```
pip install bitarray
python3 demo_amazon.py
python3 demo_wikipedia.py
```

A demo irá carregar o índice e emitir um prompt perguntando qual consulta você gostaria de realizar. Junto com os documentos retornados pelas consultas aos índices, também retornamos os tempos de processamento e a quantidade de falsos positivos (no caso do índice probabilístico).

```
$ python3 demo_amazon.py
Type your query below:
guitar
Signature Index Results:
Query: guitar
Query processing time: 0.089524s
False positives: 0.0%
3226 match(es) out of 10261 documents.

Inverted Index Results:
Query: guitar
Query processing time: 0.000418s
```

```
3226 match(es) out of 10261 documents.

Type your query below:
guitar great product
Signature Index Results:
Query: guitar great product
Query processing time: 0.21152s
False positives: 0.0%
135 match(es) out of 10261 documents.

Inverted Index Results:
Query: guitar great product
Query processing time: 0.001457s
135 match(es) out of 10261 documents.

$ python3 demo_wiki.py

Type your query below:
soccer football
Signature Index Results:
Query: soccer football
Query processing time: 0.152316s
False positives: 0.0%
19 match(es) out of 10000 documents.

Inverted Index Results:
Query: soccer football
Query processing time: 0.0001s
19 match(es) out of 10000 documents.

Type your query below:
this is not and

Signature Index Results:
Query: this is not and
Query processing time: 0.279467s
False positives: 0.0%
573 match(es) out of 10000 documents.

Inverted Index Results:
Query: this is not and
Query processing time: 0.005441s
573 match(es) out of 10000 documents.
```

4 Resultados

4.1 Falsos Positivos

Com os parâmetros usados não consegui encontrar uma consulta que retornasse falsos positivos. Usei três funções de hash e percebi que reduzir o número de bits do filtro aumentava o número de falsos positivos, mas não influenciava significativamente no tempo de processamento da consulta. Por isso optei por usar um vetor de bits um pouco maior, grande o suficiente para não retornar quase nenhum falso positivo.

De qualquer forma, falsos positivos não são realmente um problema para recuperação na web, visto que o objetivo de quem realiza a consulta não costuma ser encontrar os documentos que contém exatamente todos os termos da consulta. No caso do Bing, o BitFunnel é usado como um filtro upstream barato para reduzir o número de documentos que são enviados ao oráculo, que será responsável pelo rankeamento dos documentos da consulta - mas que possui um custo de processamento muito mais elevado.

4.2 Tempo de Processamento das Consultas

O tempo de processamento das consultas com o nosso índice de assinaturas foi consideravelmente mais alto que o do índice invertido. Isso já era esperado, pois ele precisa percorrer todos os documentos da coleção para cada bit na consulta.

Esta é uma das desvantagens que desmotivou a pesquisa na área até então. Os resultados do BitFunnel foram tão positivos porque foram implementadas algumas otimizações.

4.3 Inovações do BitFunnel

Bit-Sliced Block Signatures: no caso do BitFunnel, cada vetor não representa apenas um único documento, mas sim a união de alguns documentos. Isso reduz consideravelmente o número de consultas necessárias, mas aumenta a quantidade de falsos positivos - o que, como vimos anteriormente, não é um grande problema.

Frequency Conscious Signatures: um dos problemas da nossa abordagem é o uso ineficiente de memória, visto que a frequência dos termos varia drasticamente, com alguns inclusos em todos os documentos e, outros, em apenas um ou dois. A solução encontrada pelos engenheiros do BitFunnel foi em considerar a frequência relativa dos termos para decidir o número de hashes necessários para mapear cada termo. Os termos mais raros precisam ser mapeados em mais bits, enquanto os mais comuns são mapeados para menos posições.

Sharding by Document Length: filtros de Bloom possuem um outro problema: todos os documentos precisam ter a mesma configuração (o número de bits precisa ser o mesmo e todos precisam usar o mesmo esquema de hashing). Para evitar falsos positivos é preciso manter um número de bits compatível com o documento mais longo da coleção, mesmo que documentos com esse tamanho sejam raros. No BitFunnel os documentos são separados pelo comprimento e indexados em shards diferentes, garantindo que todos os documentos de um shard terão tamanhos similares e não haverá desperdício de memória.

Referências

1. Bob Goodwin, Michael Hopcroft, Dan Luu, Alex Clemmer, Mihaela Curmei, Sameh Elnikety, Yuxiong He (2017) BitFunnel: revisiting signatures for search, SIGIR'17
2. Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. 1998. Inverted Files versus signature files for text indexing. ACM Transactions on Database Systems (TODS)23,4(1998), 453-490
3. Michael Hopcroft (2017) BitFunnel: Revisiting Signatures for Search in-depth talk, https://www.youtube.com/watch?v=1-Xoy5w5ydM
4. https://pypi.org/project/bitarray/
5. Julian McAuley, Amazon Product Data, UCSD, 5-core Musical Instruments Dataset http://jmcauley.ucsd.edu/data/amazon/
6. http://fulmicoton.com/tantivy-files/wiki-articles-1000.json