Busca por similaridade em índice invertido utilizando o paradigma Map-Reduce

Paulo V. M. Cardoso, Sergio L. S. Mergen

Curso de Ciência da Computação – Universidade Federal do Santa Maria (UFSM)
Santa Maria – RS – Brazil

{pcardoso, mergen}@inf.ufsm.br

Abstract. *In the Information Retrieval area, inverted indexes are commonly used to do object query. The similarity search can be a powerful strategy, once it allows the retrieval of objects that would not be found with a exact match search. However, the amount of data and the search complexity may lead to a prohibitive processing cost. This work explore the possibility of distributing this search type process by using Map-Reduce paradigm. The paper propose search experiments and compares the performance between distributed and pseudo-distributed approaches.*

Resumo. *Na área de Recuperação de Informação, é comum o uso de índices invertidos para realizar a busca por objetos. A consulta por similaridade pode ser uma estratégia poderosa, pois consegue recuperar objetos que poderiam não ser encontradas em buscas por equivalência. Porém, a quantidade de objetos e complexidade da busca podem tornar proibitivo o custo desse processamento. Esse trabalho explora a possibilidade de distribuição de processamento, através do modelo de programação Map-reduce, para o tipo de busca apresentado. O artigo propõe um mapeamento para consulta e faz uma comparação entre as abordagens distribuída e pseudo-distribuída.*

1. Introdução

A Recuperação de Informação é uma área de pesquisa que visa a busca de objetos armazenados a partir de consultas, em que são informadas palavras chave (termos) que caracterizam os objetos que se pretende acessar [Korfhage 2008]. A eficiência na busca de objetos depende de vários aspcetos, que compreendem desde a definição de uma infraestrutura que suporte a computação até o uso de estruturas de dados adequadas que realizem a indexação dos objetos.

Uma estrutura bastante utilizada para este princípio é chamada de índice invertido, em que uma entrada do índice leva à uma lista de objetos relacionados, baseado em uma estrutura de pares chave-valor. Esta estrutura é atraente quando se tem um sistema de busca que é formulado por uma lista de palavras chave, uma vez que os objetos alvo são facilmente encontrados na estrutura do índice a partir de seus termos.

Consultas compostas por termos podem ser usadas para a realização de buscas por equivalência ou por similaridade. O primeiro caso pode não encontrar objetos relacionados à busca, pois faz consultas apenas por igualdade. Em contrapartida, a busca por similaridade apresenta uma estratégia mais poderosa, recuperando mais objetos, porém tem seu custo elevado. Neste caso, de acordo com a computação da

similaridade e a complexidade utilizadas, pode ser necessário recorrer à técnicas de distribuição de processamento para que se obtenha um desempenho satisfatório.

Uma dessas técnicas é o *framework* Map-reduce [Dean and Ghemawat 2008], que foi proposto para o desenvolvimento de algoritmos distribuídos. O Map-reduce é baseado em duas funções principais: o *map* e o *reduce*. O *mapper* foi projetado para que os dados de entrada pudessem ser divididos com base em um índice, podendo-se processar as divisões de forma paralela e distribuída. Já o *reducer* tem como objetivo receber pares chave-valor das etapas anteriores, processando-os e agrupando os resultados na saída.

A abordagem de mapeamento e redução chama a atenção pela semelhança ao conceito de índice invertido, pois a comunicação entre cada processo do *framework* é feita com listas de pares chave-valor, assim como é estruturado o índice citado.

O ambiente mais indicado para a execução de algoritmos que utilizam o modelo Map-Reduce são os *clusters*, em que nós são definidos por diversas máquinas que intera-gem entre si para a execução de uma tarefa. A *clusterização* permite que processamentos que demandam alto desempenho sejam possíveis mesmo se os recursos computacionais são limitados, através da distribuição de tarefas entre mais de um recurso computacional.

Neste contexto, o objetivo deste trabalho é analisar de que forma o Map-reduce se comporta para realizar a distribuição de processamento de um problema de busca por similaridade sobre índices invertidos. O trabalho consiste em propor um algoritmo de ma-peamento para esse problema e demonstrar, através de experimentos, situações em que o uso do processamento distribuído pode ser otimizado para a melhora do seu desempenho, além de comparar com uma abordagem em que o *framework* é utilizado por apenas 1 nó.

2. Mapeamento

O mapeamento proposto para o problema de busca em um índice invertido *I* usando o modelo Map-Reduce tem como configuração de entrada o próprio arquivo do índice. Desta forma, o *framework* irá realizar uma tarefa *map* (*map task*) para cada par chave-valor contido na estrutura do índice. Esta abordagem foi escolhida por ter uma característica escalável, uma vez que a inserção de mais nós no *cluster* possibilita uma maior distribuição naturalmente.

Todo *mapper* recebe uma lista de entradas *E* do índice *I* e, então, executa uma *map task* para cada par chave-valor contido no seu *input*, em que é processado um único par chave-valor por vez, contendo um termo C_i e uma lista de objetos O_i^j associados. Conforme é mostrado na Tabela 1, a saída dessa etapa consiste em novos pares chave-valor contendo os objetos *O* recebidos e o score calculado através da similaridade entre *C* e os termos da consulta (*keywords*). Na etapa de redução os escores são somados e levados à saída do algoritmo como um par chave-valor constituído da soma de sua similaridade e os objetos relacionados àquele escore.

Tabela 1. Entradas e saídas das etapas do mapeamento proposto

Etapa	Entrada	Saída
mapper	Lista de E	$[\{O_1, Score_i\}, \{O_2, Score_i\}, ..., \{O_n, Score_i\}]$
reducer	*list*({*O, Lista de scores*})	*list*({*sum*(*scores*), *Lista de O*})

O mapeamento proposto emite as similaridades entre termos mesmo se o nível de equivalência é baixo. Para evitar isso, foi criado uma variação do algoritmo em que apenas as similaridades acima de um determinado limite (*threshold*) são consideradas, de forma que o Map-reduce deixe de trabalhar com informações irrelevantes para a consulta.

3. Experimentos

Para os experimentos, utilizou-se a versão 2.7.2 do Apache Hadoop para a implementação do Map-reduce. Foi configurado um *cluster* contendo 1 nó mestre e outras duas máquinas escravas, todos com configurações idênticas: processador de dois núcleos com frequência de 2.2 GHz, 7,5GB de memória RAM e 50GB de disco. A função de similaridade utilizada é baseada no algoritmo de Levenshtein, que calcula o número de edições necessárias para transformar um elemento textual em outro [Ristad and Yianilos 1998].

O índice invertido foi criado sobre a coleção de objetos do BioID [Cardoso et al. 2016], um ambiente colaborativo para a manutenção e catalogação de dados de espécies biológicas. Nesse ambiente, os objetos são definidos por espécies que possuem uma série de propriedades indexáveis (atributos). Um objeto pode conter os atributos *Reino* e *Classe* e seus respectivos valores *Animalia* e *Mammalia*, sendo que os valores também são considerados atributos. A base de dados do BioID conta com 222.120 espécies e uma média de 4 atributos por cada objeto. O número de objetos, porém, pode crescer conforme a utilização do sistema.

Foram criados testes para dois cenários diferentes: utilizando o Hadoop de forma pseudo-distribuída (usando apenas uma máquina) e, por fim, testando o mapeamento de forma distribuída entre os nós do *cluster*. Para o primeiro caso, apenas o nó mestre foi utilizado, enquanto que o cenário distribuído envolveu as três máquinas no papel de escravos, sendo que o nó mestre também desempenhou a função de gerente da aplicação.

Nos primeiros testes, a busca foi realizada com variação no número de termos utlizados na consulta. Foram submetidas consultas com 1, 2, 5 e 10 palavras chave. A Figura 1 mostra o comportamento nos dois cenários configurados através da busca na base de dados replicada em 20 vezes. Pode-se notar que não existe uma grande diferença de desempenho entre a distribuição e a execução em apenas um nó, mesmo que o método distribuído se mostre vantajoso em consultas com mais de um termo.

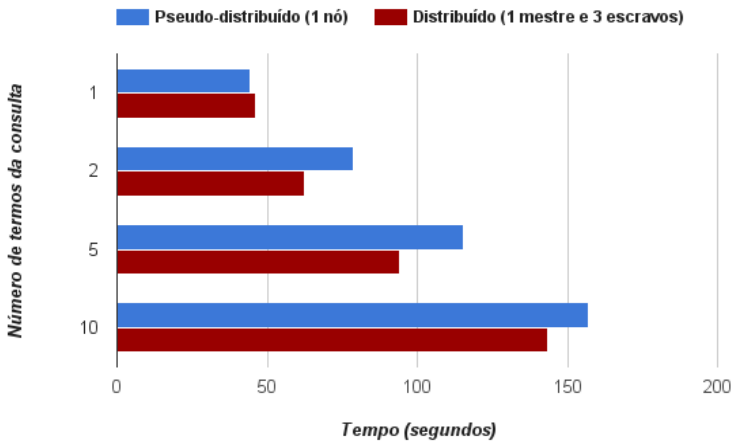

Figura 1. Desempenho do algoritmo de busca variando-se o número de termos

Para o próximo teste, foi configurada a filtragem de relações de similaridade consideradas baixas através de um *threshold* mínimo. Foram testados os níveis de 0% (sem *threshold*), 25%, 50%, 80% e 90%, conforme exibido na Figura 2, usando 5 termos uma consulta na mesma base dos testes anteriores. O tempo de execução cai de forma significativa a partir do threshold limitado em 25%, mantendo o desempenho dos cenários usados bastante próximos na medida em que esse filtro aumenta.

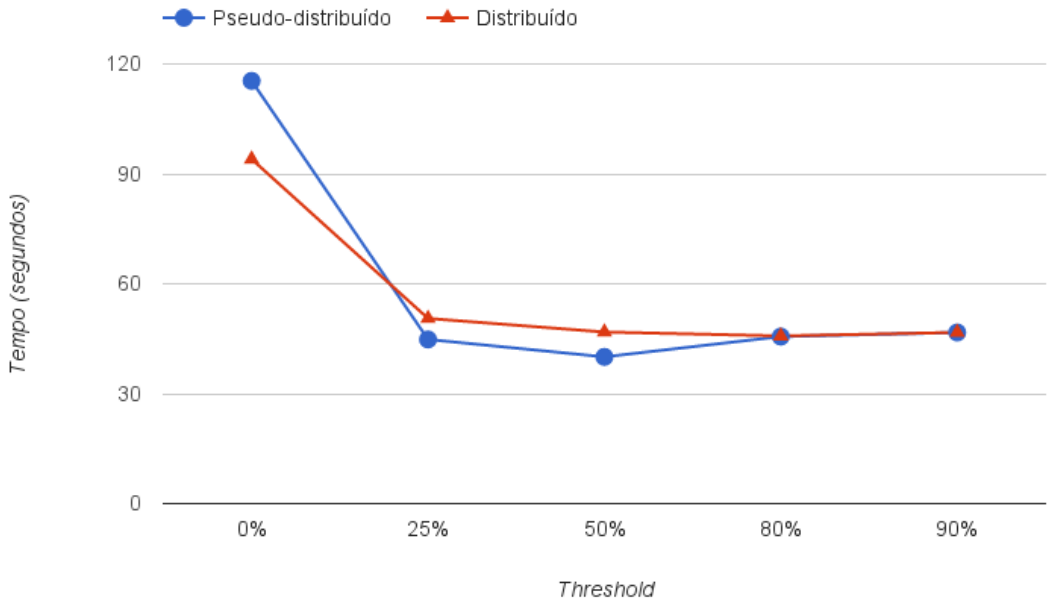

Figura 2. Busca com 5 termos utilizando diferentes níveis de threshold

4. Considerações finais

Este trabalho mostrou uma análise de desempenho para um problema de busca em índice invertido utilizando uma abordagem distribuída através do Map-reduce. Pode-se notar que uma solução distribuída para este tipo de problema é vantajosa em relação à uma aplicação pseudo-distribuída. Esse desempenho pode se tornar ainda mais eficiente com a inserção de novos nós no *cluster*.

Além disso, a aplicação de um *threshold* se mostra útil para incrementar o desem-penho da aplicação, uma vez que este procedimento diminui o volume de informações processadas pelo *framework*, assim como diminui o tamanho dos *outputs* de *mappers* e *reducers*.

Referências

Cardoso, P. V., F., F. F., and L.S., M. S. (2016). Using active mediators and passive extractors inside materialized data integration systems. Congresso da Sociedade Brasileira de Computação - CTIC, 35:501–510.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107–113.

Korfhage, R. R. (2008). Information storage and retrieval.

Ristad, E. S. and Yianilos, P. N. (1998). Learning string-edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(5):522–53220(5):522–532.