Análise de ferramentas que implementam o paradigma MapReduce em um problema de Recuperação de Informação

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

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

{pcardoso, mergen}@inf.ufsm.br

Resumo. *Na área de Recuperação de Informação, a busca por similaridade representa uma relevante estratégia de consulta. Porém, a complexidade do cálculo da similaridade pode tornar proibitivo o custo desse processamento, especialmente quando o volume de dados considerado é alto. Esse trabalho explora a o uso do MapReduce como forma de reduzir o tempo de processamento, através da comparação do Apache Hadoop e Apache Spark.*

1. Introdução

A Recuperação de Informação é uma área de pesquisa que visa a busca de objetos armazenados, a partir de consultas compostas por palavras chave (termos) que caracterizam os objetos que se pretende acessar [Korfhage 2008]. Uma estrutura regularmente utilizada para esse fim é chamada de índice invertido, em que uma entrada do índice leva à uma lista de objetos relacionados. Consultas compostas por termos podem ser usadas para a realização de busca por equivalência ou por similaridade. A busca por similaridade é uma estratégia mais poderosa, uma vez que permite recuperar objetos mesmo quando não existe uma correspondência exata com os termos pesquisados. No entanto, a complexidade envolvida no cálculo da similaridade pode aumentar consideravelmente o tempo de execução. Para grandes repositórios de objetos, o alto custo pode inviabilizar esse tipo de busca.

Uma forma de melhorar o desempenho é através de técnicas que distribuem o processamento em um *cluster*. Um modelo de programação usado para solucionar problemas envolvendo grandes quantidades de dados é o MapReduce [Dean and Ghemawat 2008]. A arquitetura do MapReduce possui duas etapas principais: mapeamento (*map*) e redução (*reduce*). Enquanto a primeira etapa divide a entrada no *cluster* e realiza um processamento, a etapa de redução foca no reagrupamento dos dados. Exemplos de *frameworks* de distribuição baseados neste paradigma são o Apache Hadoop e o Apache Spark. Apesar de ambos fornecerem suporte às funções de mapeamento e redução, existem diferenças conceituais que levam essas abordagens a apresentarem comportamentos diferentes.

O paradigma MapReduce já foi usado para problemas relacionados à recomendação de documentos [Elsayed et al. 2008] [Lin 2009]. No entanto, não foram encontrados trabalhos que investigassem o seu uso para a busca por similaridade utilizando índices invertidos. Assim, o objetivo deste trabalho é analisar como as duas ferramentas mencionadas se comportam nesse caso específico. A partir dos resultados alcançados, serão analisadas as diferenças entre as duas abordagens.

2. Mapeamento da busca por similaridade

A definição de busca por similaridade usada neste trabalho foca na recuperação dos objetos mais similares a uma determinada consulta, usando a estrutura de índice invertido para organizar o acesso aos objetos. Neste caso, considera-se que: i) a consulta seja composta por termos, ii) os objetos são compostos por propriedades, iii) o índice é formado por pares chave-valor contendo as entradas (propriedades) e os objetos relacionados.

A similaridade entre um objeto e uma consulta é obtida pela soma das similaridades entre cada um dos termos da consulta e cada uma das propriedades do objeto. O escore de similaridade entre termo/propriedade é normalizado entre zero e um. Além disso, um limiar (*threshold*) é aplicado para indicar a partir de que valor um escore deve ser usado no cálculo.

O cálculo da similaridade de uma consulta e todos os objetos indexados foi implementado através das abordagens centralizada (sequencial) e distribuída. Na primeira, todo o processo é realizado de forma sequencial, sem qualquer tipo de paralelismo. Já na abordagem distribuída, o paradigma MapReduce foi usado. Com MapReduce, a etapa de mapeamento recebe como entrada uma parte do índice, ou seja, uma lista de propriedades e seus respectivos objetos. A partir dessa lista, é calculada a similaridade entre cada termo da consulta e cada uma das propriedades. Os escores acima do limiar são enviados para a etapa de redução, sendo que a chave emitida é o objeto associado à propriedade e o valor emitido é o escore encontrado. Na etapa de redução, realiza-se a soma dos escores de cada objeto. Como a saída da redução obedece a ordem das chaves, os objetos mais similares são retornados antes.

2.1. Ferramentas utilizadas

O Hadoop oferece uma das mais famosas implementações do paradigma MapReduce, e é baseado em dois módulos: o sistema de arquivos distribuído (HDFS) e o gerenciador de recursos (YARN). Esse *framework* é conhecido por sua grande escalabilidade e tolerância a falhas. Porém, apenas uma etapa de mapeamento e redução pode ser usada em cada processo (*job*)[1].

O Apache Spark possibilita uma maior simplicidade de desenvolvimento através do uso de um grafo acíclico dirigido (DAG) para definir o encadeamento de funções de mapeamento e redução. No Hadoop, esse encadeamento só seria possível através da configuração de diferentes *jobs*. Além disso, este *framework* utiliza a memória principal para a comunicação entre o mapeamento e a redução (através de uma estrutura chamada de *Resilient Distributed Dataset*). Já no Hadoop, arquivos intermediários devem ser usados para guardar a saída gerada pelo mapeamento. A influência dessas diferenças no processamento de busca por similaridade será analisada na próxima seção.

3. Experimentos

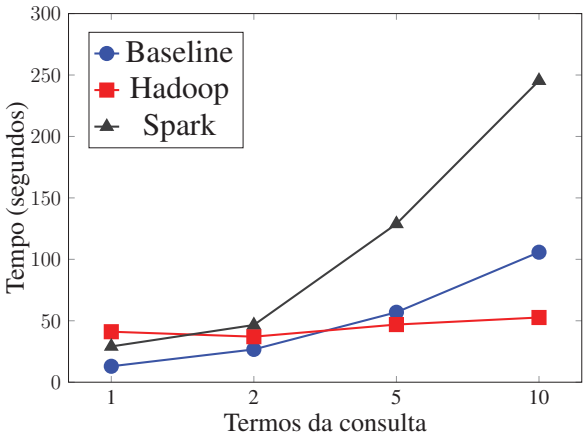
Os experimentos foram submetidos em um ambiente de *cluster* com 3 nós, sendo 1 mestre (que também atuou como trabalhador) e outros 2 nós trabalhadores, todos com configurações idênticas: processador 2-core (2.4 GHz) e 7,5 GB de RAM. O Hadoop rodou na versão 2.7.2 e o Spark na versão 2.0.2. O índice invertido foi criado com base

[1] na verdade, existe suporte para que a saída do mapeamento seja preprocessada antes do envio à redução (através de uma etapa chamada de *combiner*)

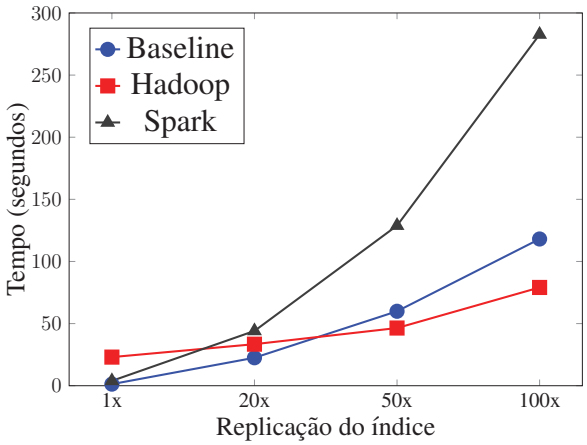
no repositório do sistema BioID [Cardoso et al. 2016], que armazena informações sobre cerca de 50.000 espécies de seres vivos, classificados de forma taxonômica através de seus atributos. O índice criado possui 27.000 propriedades e ocupa 1,9 MB em disco. A função de similaridade usada foi o algoritmo de distância de edição de Levenshtein.

Foram criados testes para o cenário centralizado (*Baseline*) e cenários distribuídos envolvendo as ferramentas Hadoop e Spark. Para o primeiro caso, apenas o nó mestre foi utilizado, enquanto que o cenário distribuído envolveu as três máquinas. O escalonador do Hadoop foi alterado para uma abordagem de distribuição justa, pois esta aproveita melhor os recursos do *cluster*. O resultado de cada teste é constituído da média do tempo entre 20 repetições feitas.

Inicialmente, a busca foi realizada com variação no número de termos da consulta e no tamanho da base de dados, como mostram as Figuras 1(a) e 1(b). Pode-se notar que apenas o Hadoop apresenta uma alta escalabilidade nos dois cenários. Em relação ao *Baseline*, o Spark consegue resultados satisfatórios se aplicado em bases pequenas, enquanto o Hadoop ganha em tempo de execução quando mais objetos são considerados.



(a) Cenário com variação do número de termos em ums busca com índice aumentado em 50 vezes e 50% de threshold



(b) Cenário com aumentos na base de dados em uma busca com 5 termos e 50% de *threshold*

Figura 1. Escalabilidade das ferramentas

A fim de aumentar o nível de distribuição das aplicações, as ferramentas foram configuradas para criarem um número maior de divisões da entrada. Para este cenário, o índice foi fragmentado em 3, 6 e 12 fatias (*splits* no Hadoop e *executors* no Spark). Como mostra a Figura 2, o aumento de divisões proporciona um melhor desempenho, porém em diferentes circunstâncias. No Spark, o número de *executors* define a quantidade de módulos de execução que rodam em paralelo através dos nós. Por outro lado, o número de *splits* do Hadoop não define, necessariamente, uma paralelização completa, já que o escalonador justo utiliza uma fila FIFO de tarefas para cada nó, gerando sobrecarga quando o número de nós for menor do que o número de fatias.

Para sumarizar, a Tabela 1 compara as ferramentas dentro do contexto investigado. Apesar de o Spark ter um processamento baseado em memória primária, a escalabilidade é baixa para o problema de busca por similaridade. Os resultados sugerem que o custo relativo ao gerenciamento dos RDDs supera o custo das operações de entrada e saída ne-

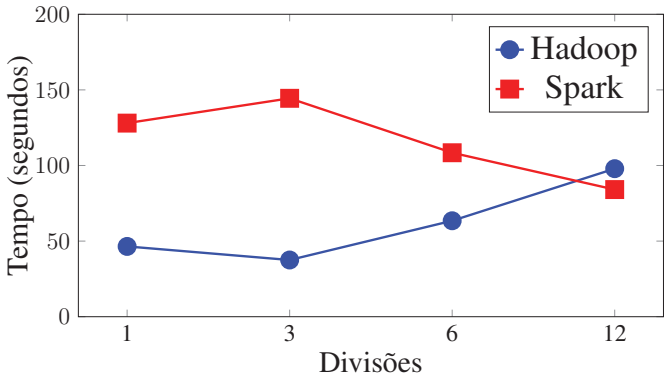


Figura 2. Relação entre o desempenho dos mapeamentos e o tamanho do índice

cessários para o Hadoop. Além disso, é importante observar que os fatores de distribuição (divisão) usados pelas ferramentas (*splits* e *executors*) tem significados distintos. Compreender essas diferenças é essencial para que a configuração escolhida seja otimizada.

Cenário	Apache Hadoop	Apache Spark
Escalabilidade	Altamente escalável	Fracamente escalável
Divisões	Melhor até o mesmo número de nós	Melhor com mais divisões
Requisições online	Não	Com bases pequenas

Tabela 1. Resumo dos resultados das ferramentas nos cenários testados

4. Considerações finais

De modo geral, os tempos observados demonstram que as soluções testadas não são aconselháveis para requisições em tempo real, em que uma resposta para a busca por similaridade deve ser consumida de imediato. O uso de diferentes métricas e estruturas de dados podem ser explorados para a aplicabilidade da solução. Por outro lado, também se faz necessária uma investigação do comportamento das ferramentas utilizadas em problemas onde o tempo de execução não é um fator crítico. Um exemplo é a busca por similaridade em bases de dados mais complexas, como aquelas usadas por sistemas de reconhecimento facial. Neste caso, a opção por ferramentas de distribuição pode ser determinante para a redução do tempo de resposta.

Referências

Cardoso, P. V., F., F. F., and L.S., M. S. (2016). Using active mediators and passive extractors inside materialized data integration systems. *CSBC - 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.

Elsayed, T., Lin, J., and Oard, D. W. (2008). Pairwise document similarity in large collections with mapreduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies*, pages 265–268.

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

Lin, J. (2009). Brute force and indexed approaches to pairwise document similarity comparisons with mapreduce. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 155–162. ACM.