PageRank Algorithm in MapReduce
Projeto Final

Aluno de Doutorado: Antônio de Abreu Batista Júnior
Universidade Federal do ABC
antonio.batista@ufma.br

Abstract: To analyze the massive data sets, a distributed computing framework is needed on top of a cluster of servers. MapReduce is the most popular distributed framework for Big Data processing. In the work, I discuss and present the PageRank algorithm over distributed system using MapReduce in haskell.

1. Introdução

PageRank é uma medida da qualidade de páninas web baseada na estrutura do grafo web. Este grafo pode ser caracterizado como um grafo grande e complexo. Processar grafos grandes usando computação tradicional não é possível. Atualmente, a única abordagem escalável é usar computação distribuída.

MapReduce é um framework distribuído para processar dado em grande escala sobre clusters de máquinas, além de um modelo de programação para expressar computações distribuídas em conjuntos de dados massivo.

Neste trabalho, eu discuto e apresento uma implementação escalável, em haskell, do algoritmo pagerank em MapReduce [1] usando um framework implementado em haskell e ensinado nas aulas da disciplina de Inteligência na Web e Big Data [2]. O framework usa a força da avaliação pregisosa e a paralelização de listas como base de seu funcionamento. O framework pode ser adaptado sem grande esforço para outros problemas exigindo somente pequenas alterações. A seguir é dado um background matemático sobre o pagerank, e logo em seguida, uma implementação MapReduce proposta por [1] que inspirou este trabalho.

2. background matemático

2.1. PageRank

O vetor *PageRank PR* é definido sobre um grafo direcionado $G = (V, E)$. Cada nó v no grafo está associado com um valor *PageRank* PR^{i+1} v. O valor inicial de cada nó é $\frac{1}{N}$ e cada nó v atualiza seu valor *PageRank* iterativamente pela equação 2. A Tabela 1 mostra a notação usada neste trabalho.

$$demp\ x = \frac{1-d}{N} + d \times x \tag{1}$$

$$PR^{i+1}\ m = demp\ (\sum_{n\ |\ m \in out\ n} \frac{PR^i\ n}{C\ n}) \tag{2}$$

Table 1. Notação usada na discussão teórica.

m e n	variáves variando sobre os nós do grafo
$out\ m$	um multiset de links de saída (cada link representado pelo nó alvo) do nó m
$C\ m$	a cardinalidade de $out\ m = \#out\ m$

[1] Os códigos dessa implementação estão disponíveis em https://github.com/antoniodeabreu/-BIGDATA2017/tree/master/ProjetoMapReduce
[2] O framework MapReduce está disponível em https://github.com/folivetti/BIGDATA/blob/master/MapReduce/Code.md

2.2. *PageRank em MapReduce*

A computação em MapReduce de PR^{i+1} a partir de PR^i é definida pelas funções *mapper* e *reducer*. Em que a tupla $(m,\ (rank, out\ m))$ representa a entrada e a saída da função. O primeiro elemento da tupla representa a entrada da função e o segundo elemento a sua saída. PR^i é a entrada do *mapper* e PR^{i+1} a saída do *reducer*. A seguir está a especificação destas funções:

1. $mapper\ (m,\ (p,ns)) = [(n, (\frac{p}{\#ns}, [\,])) \mid n \leftarrow ns] +\!\!+ [(m, (p, [\,])) \mid ns = [\,]] +\!\!+ [(m, (0, ns))]$
2. $reducer\ (m,\ xs) = (m,\ (demp(sum\ ps),\ concat\ ns))$
   $where\ ps = map\ fst\ xs,\ ns = map\ snd\ xs$

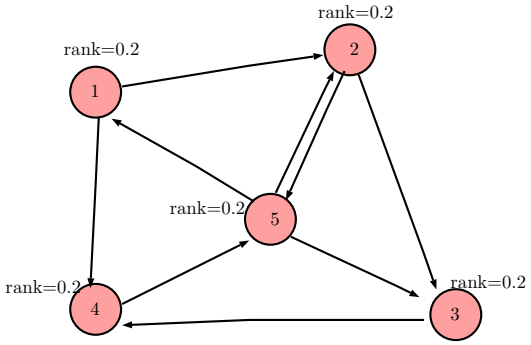

Fig. 1. Grafo de brinquedo: com o pagerank PR^0 m de cada nó m na iteração $i = 0$.

A seguir é dado um exemplo de uso destas funções. Em cada iteração são calculados os valores pagerank de todos os nós do grafo Fig. 2.2. Para exemplificar suponha que vamos calcular a iteração $i = 1$ e que os dados do grafo representam a iteração $i = 0$. A função *mapper* é aplicada para cada entrada (nó do grafo). O framework MapReduce cuidará de agrupar as saídas dos *mappers* com as mesmas chaves em uma mesma máquina na qual roda uma função *reducer*. Esta função atua em cada entrada única (nó do grafo). Estes passos são mostrados a seguir:

1. mapper.
   $(1, (0.2, [2,4])) \overset{mapper}{\rightarrow} (2, (0.1, [])),\ \ (4, (0.1, [])),\ \ (1, (0, [2,4]))$
   $(2, (0.2, [5,3])) \overset{mapper}{\rightarrow} (3, (0.1, [])),\ \ (5, (0.1, [])),\ \ (2, (0, [5,3]))$
   $(3, (0.2, [4])) \overset{mapper}{\rightarrow} (4, (0.2, [])),\ \ (3, (0, [4]))$
   $(4, (0.2, [5])) \overset{mapper}{\rightarrow} (5, (0.2, [])),\ \ (4, (0, [5]))$
   $(5, (0.2, [1,2,3])) \overset{mapper}{\rightarrow} (1, (0.066, [])),\ \ (2, (0.066, [])),\ \ (3, (0.066, [])),\ \ (5, (0, [1,2,3]))$
2. O agrupamento dos resultados do mapper (feito pelo framework MapReduce):
   $(1, [(0, [2,4]),\ \ (0.066, [])])$
   $(2, [(0.1, []),\ \ (0, [5,3]),\ \ (0.066, [])])$
   $(3, [(0.1, []),\ \ (0, [4]),\ \ (0.066, [])])$
   $(4, [(0.1, []),\ \ (0.2, []),\ \ (0, [5])])$
   $(5, [(0.1, []),\ \ (0.2, []),\ \ (0, [1,2,3])])$
3. reducer
   $(1, [(0, [2,4]),\ \ (0.066, [])]) \overset{reducer}{\rightarrow} (1, (demp\ (0.066), [2,4]))$
   $(2, [(0.1, []),\ \ (0, [5,3]),\ \ (0.066, [])]) \overset{reducer}{\rightarrow} (2, (demp\ (0.166), [5,3]))$

$$(3,[(0.1,[]),\ (0,[4]),\ (demp\ (0.066),[])]) \overset{reducer}{\rightarrow} (3,(demp\ (0.166),[4]))$$

$$(4,[(0.1,[]),\ (0.2,[]),\ (0,[5])]) \overset{reducer}{\rightarrow} (4,(demp\ (0.3),[5]))$$

$$(5,[(0.1,[]),\ (0.2,[]),\ (0,[1,2,3])]) \overset{reducer}{\rightarrow} (5,(demp\ (0.3),[1,2,3]))$$

3. Adaptação do framework MapReduce

1. Pré-processamento

   transforma o arquivo de entrada em uma nova entrada. A saída do pré-processamento é uma lista contendo as saídas da função *mapper* aplicada a cada nó do grafo. Como entrada para o pagerank foi utilizado o grafo web da Universidade de stanford [3].

2. pagerank em MapReduce

```
pagerank :: Int -> [(Int, (Double, [Int]))] -> M.HashMap Int (Double, [Int])
pagerank n nl = reducer myreducer mapped
 where
  mapped      = map mapfun workers `using` parList rdeepseq
  mapfun      = mapper myreducer mymapper
workers = chunksOf n nl
```

3. mymapper definida pelo usuário

```
mymapper :: (Int, (Double, [Int])) -> (Int, (Double, [Int]))
mymapper m = m
```

4. myreducer definida pelo usuário

```
myreducer :: (Double, [Int])-> (Double, [Int]) -> (Double, [Int])
myreducer (m1, xs1) (m2, xs2) = (demp (m1 + m2), xs1 ++ xs2)
```

4. Conclusões

O framework disponível em aula foi adaptado com sucesso para o pagerank. A implementação tornou o algoritmo não só escalável, mas também otimizou a utilização dos recursos de hardware (memória e processador) de cada máquina graças a execução em paralelo dos pedaços de dado e os seus descates quando não mais necessários.

5. References

[1] FOKKINGA, M. Mapreduce formulation of pagerank. Unpublished Technical Report, 2010.

[3] O grafo pode ser baixado neste endereço https://snap.stanford.edu/data/web-Stanford.html