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 implement the PageRank algorithm over distributed system using MapReduce.

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 complexo e grande. 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.

Este trabalho discute o projeto e a implementação do algoritmo pagerank em MapReduce. Uma implementação em haskell é dada a fim de demonstrar o correto funcionamento deste algoritmo [1]. Para uma prova formal da corretude do algoritmo consulte [1]

2. 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 $out\ m = \#out\ m$

3. PageRank em MapReduce

A computação de PR^{i+1} a partir de PR^i pode ser formulada como uma computação MapReduce. Em que, PR^i é a entrada do $mapper$ e PR^{i+1} a saída do *reducer*. A especificação destas funções são dadas a seguir, em que o conjunto de pares $(entrada, saida) = (m,\ (rank, out\ m))$ representam a entrada da função $mapper$ e a saída da função *reducer*.

[1] https://github.com/antoniodeabreu/-BIGDATA2017/tree/master/ProjetoMapReduce

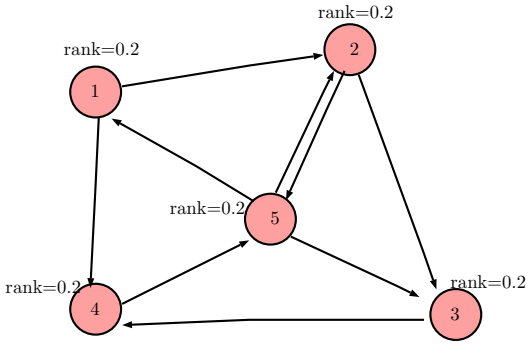

Fig. 1. Grafo de brinquedo: com o pagerank PR^0 m de cada nó m na iteração $i = 0$.

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

Para calcular os valores pagerank de todos os nós do grafo de brinquedo da Fig. 3, na iteração seguinte $i = 1$, basta mapear a função *mapper* para cada entrada (nó do grafo). O framework MapReduce cuidará de agrupar as saídas dos *mappers* com as mesmas chaves em uma mesma máquina. Para só, então, mapear a função *reducer* para cada entrada única. 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
   $(1, [(0, [2,4]),\ (0.066, [])]) \overset{reducer}{\rightarrow} (1, (demp\ (0.066), [2,4]))$
   $(2, [(0.1, []),\ (0, [5,3]),\ (0.066, [])]) \overset{reducer}{\rightarrow} (2, (demp\ (0.166), [5,3]))$
   $(3, [(0.1, []),\ (0, [4]),\ (demp\ (0.066), [])]) \overset{reducer}{\rightarrow} (3, (demp\ (0.166), [4]))$
   $(4, [(0.1, []),\ (0.2, []),\ (0, [5])]) \overset{reducer}{\rightarrow} (4, (demp\ (0.3), [5]))$
   $(5, [(0.1, []),\ (0.2, []),\ (0, [1,2,3])]) \overset{reducer}{\rightarrow} (5, (demp\ (0.3), [1,2,3]))$

4. Considerações finais

Foi feita uma implementação na linguagem de programação *haskell* do algoritmo *pagerank* em *MapReduce*. A fim de verificar o correto funcionamento das tarefas *mapper* e *reducer*, algumas tarefas específicas do *framework MapReduce* foram simuladas, por exemplo a tarefa de agrupar tuplas emitidas pelos *mappers* com a mesma chave que caberia ao *framework* é simulada por um método em haskell. Por fim, esta é uma implementação didática do *pagerank* em *MapReduce*.

5. References

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