CMS/CS/EE144 **Team Kaigoo**

Miniproject : Rankmaniac

02/10/2018 Hangwen Lu, Guanya Shi, Botao Hu

1 Team Member & Work Split

- **Team Name**

  Team Kaigoo

- **Group Members**

  Hangwen Lu
  Guanya Shi
  Botao Hu

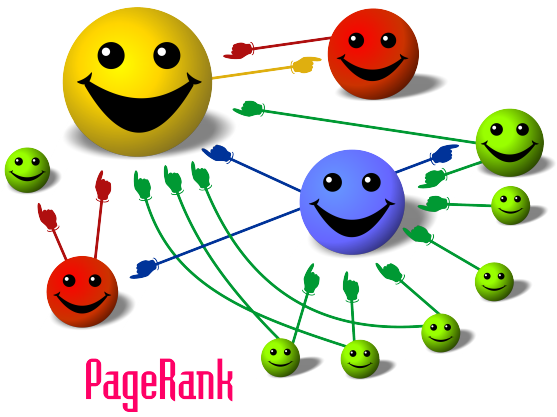
- **Division of Labor**

  Hangwen Lu: Collaborated on implementing basic pagerank algorithm to MapReduce, debugging, local test and optimization, submit models to Amazon server and write algorithms part in the report.
  Guanya Shi: Collaborated on implementing the PageRank map-reduce algorithm; designed the linear extrapolation algorithm and the top-k convergence threshold to decrease the number of iterations.
  Botao Hu: Collaborated on implementing the PageRank algorithm; built the interface of AWS MapReduce and local code; proposed the idea of extrapolation; organized the wrting of the report.

2 Introduction

- **PageRank**

  PageRank is a link analysis algorithm named after Larry Page, one of the founders of Google. This algorithm ranks each node in a web graph with the purpose of measuring its relative importance within the set [1]. The mathematical interpretation is the following.



Let's consider a web graph $G(V, E)$ (see figure above [1]) with n nodes, and d_i denotes the out-degree of node i. The transition matrix P is defined as

$$P = [p_{i,j}],$$

where

$$p_{i,j} = \begin{cases} \frac{1}{d_i}, & i \to j \\ 0, & \text{else} \end{cases}.$$

In order to make PageRank converge, a further restriction of P is defined as

$$G = \alpha P + \frac{1-\alpha}{n}\mathbf{1}_{n\times n},$$

where α is a damping factor (we typically have $\alpha = 0.85$ for best performance), and $\mathbf{1}$ is a matrix with all entries equal to 1.
Let $r_i(t)$ be the rank of node i at iteration t, and for all nodes, $r_i(0) = 1/n$. An iteration of PageRank calculation is defined as

$$r(t+1) = r(t) \cdot G.$$

It is proved that for a certain network structure with transition matrix G, there exists a unique page rank $\pi = \lim_{t\to\infty} r(t)$ such that

$$\pi = \pi G.$$

And this π reflects the network structure and we use this vector to rank nodes.
In practical use, companies like Google are more focused on the rank of important nodes, as well as the time complexity of calculation. As a result, they do not calculate the convergence version of π, instead, they iterate $r(t+1) = r(t) \cdot G$ untill some stopping condition is satisfied.

- **MapReduce**

  MapReduce is a programming model which is advantageous in processing huge data sets in parallel. This model is also good at splitting task in distributed system, and thus it is programmer friendly since programmer can jump ahead of complex system structure and only think of algorithm realization. The model contains basically three steps: map, collect, and reduce, data are passed between strps in the form of key-value pairs.
  According to the definition in Wikipedia [2], generally, mapper does filtering and sorting of big data, collector organized data from mapper to reducer, and reducer performs some summary operation and gets the result. For detail in our implementation, please refer to **Basic Algorithm** section.

3 Basic Algorithm

We implement Pagerank algorithm into two sequential MapReduce process:
PageRank step calculates and update the page ranks of each node; **Process** step determines the convergence condition and output the sorted top 20 nodes if it converges.

- **Pagerank_Map**

  Pagerank_Map reads in the pagerank of node i and outlinks j of node i, and emits the contribution of node i to outlinks j, e.g. $r_{i\to j} = \alpha * p_{i,j} * r_i$ for each key j. The graph structure is maintianed by emitting the original line as well.

  For each node i, we keep both current page rank (**CPR**) and previous page rank (**PPR**) for later convergence determination.

Algorithm 1 Pagerank Map

procedure PAGERANK_MAP
  NodeID i , Iteration, CPR, PPR, outlinks ← Parse *std.input*
  for node j in outlinks **do**
    contribution $r_{i\to j} \leftarrow \frac{\alpha * CPR}{d_i}$
    std.output (j, $r_{i\to j}$)
  std.output (NodeID, {IT, CPR, PPR, outlinks})

- **Pagerank_Reduce**

  After the sorting emitted (key, value) pairs by collector, for each node i, there are two kinds of values: $r_{j\to i}$ as the contributions from j to r_i and the original graph structure (node, iteration(**IT**), CPR, PPR, outlinks).

  In Pagerank_Reduce step, we update CPR with $\Sigma_j r_{j\to i} + (1-\alpha)$, and update PPR with CPR, output the updated (node, IT, CPR, PPR, outlinks).

Algorithm 2 Pagerank_Reduce

procedure PAGERANK_REDUCE
  NodeID i, {$r_{j\to i}$ for all j, {IT, CPR, PPR, outlinks}} ← *std.input*
  PPR ← *CPR*
  CPR ← $\Sigma_j r_{j\to i} + (1-\alpha)$
  std.output (NodeID, {Iteration, CPR, PPR, outlinks})

- **Process_Map**

  We keep this step as blank function, passing the original line as it comes in.

Algorithm 3 Process_Map

procedure PROCESS_MAP
  for line in *std.input* **do**
    std.output line

- **Process_Reduce**:

  Process_Reduce collects all data and generates top N ranked nodes. If it satisfies convergence condition, output the final top 20 ranks in form:

  FinalRank: CPR, NodeID

  otherwise, output the previous (node, {IT, CPR, PPR, outlinks}) pairs.

  The convergence condition is determined as following: The top N ranked nodes calculated from CPR and PPR are the same, where N is a tunable value for later optimization, the larger N is, the more accurate the result it but taking more iteration steps. Normally, N = 25 40 will converge very well for generating top rank 20 nodes. If iteration is > 50, it's forced to output the final results as well.

Algorithm 4 Process_Reduce

```
1: procedure PROCESS_REDUCE
2:     for line in std.input do
3:         NodeID i , IT, CPR, PPR, outlinks ← Parse line
4:         CPR_list ← append CPR
5:         PPR_list ← append PPR
6:     TopNCPR ← top N nodes in sort(CPR_list)
7:     TopNPPR ← top N nodes in sort(PPR_list)
8:     if TopNCPR and TopNPPR are same then
9:         for rank 1 to 20 do
10:            std.output (FinalRank: CPR, NodeID)
11:    else
12:        std.output (NodeID, {IT, CPR, PPR, outlinks})
```

4 Optimization

- **Convergence Condition**

  Since our final goal is to order the top 20 nodes, we used "top-k convergence condition", which means iteration would stop if the top k nodes in current PageRank are the same as the top k nodes in previous PageRank.

  Since the network follows heavy-tailed distribution, the top k nodes' page ranks are much larger and their ranks converge much faster compaired to the rest of nodes. Therefore, instead of waiting the pageranks of whole networks to converge, we only need to consider the top k nodes.

  Obviously, k is a integer equal or greater than 20, and there is a trade-off between accuracy and speed: as k increase, the number of iterations will increase but accuracy will also increase; as k decrease, the number of iterations will decrease but accuracy will become worse.

  We did some experiments locally on three different dataset ($G(n, p)$, EmailEnron and wiki-Vote [3]). For example, some testing results of EmailEnron are as following: As seen from the above table, when

Value of k	Number of Iterations	Penalty
25	6	12min
35	7	3.5min
40	7	3.5min
50	8	1min

  k is small (like 25), 6 iterations are enough for convergence but it also brings great penalty; when k is large (like 50), penalty is small but more 2 iterations are needed for convergence compared to $k = 25$. Finally we choose $k = 40$ as the convergence threshold to balance penalty and number of iterations.

- **Heapsort**

  Our convergence conditions depend on the n largest node PageRanks in all nodes. Since we have a huge number of nodes (more than 100000), heapsort is an efficient method to get the n largest PageRanks. Thus we used python function `heapq.nlargest` to get index of the n largest PageRanks.

- **Extrapolation**

As one iteration on AEM can take around 7 or more minutes, we hope to use as few iterations as possible. Thus our idea is to predict the next PageRank based on previous iterations' PageRanks, in which we can "skip" some iterations to save time.

Since we want to predict some iterations based on previous ones, we need to study how PageRanks change as the number of iterations increases. To do this, we recorded PageRanks of the top 6 nodes in EmailEnron dataset, and the results are in the following figure: From the results, we can find that

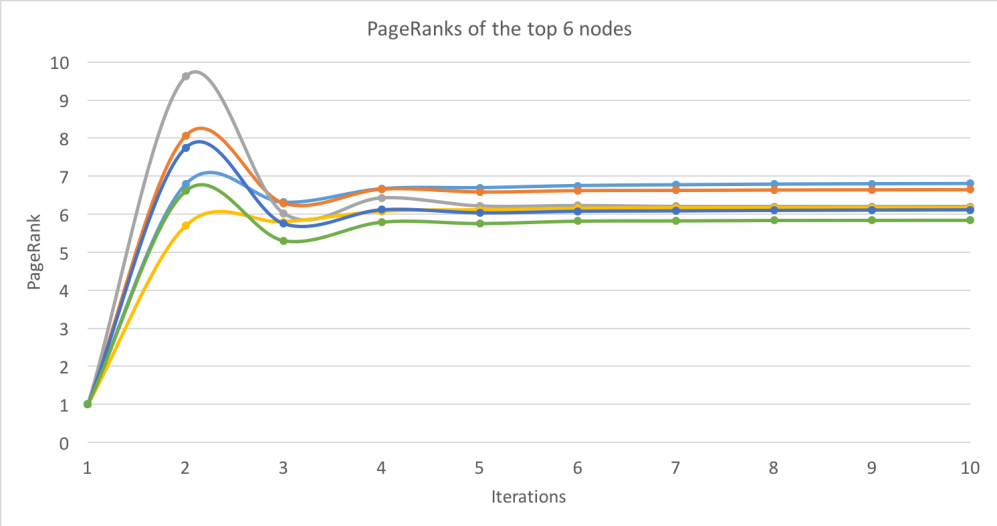


Figure 1: PageRanks of the top 6 nodes

after 4 iterations, PageRanks of these nodes change slowly and almost linearly. Thus, starting from the 4th interation, we used "linear extrapolation", which means after each normal iteration, current PageRanks and previous PageRanks of all the nodes would be predicted linearly:

$$\text{CPR} = \text{CPR} + \alpha(\text{CPR} - \text{PPR}),$$

where CPR is current PageRank, PPR is previous PageRank and α is constant to determine how much the PageRank is predicted.

We find that if α is big (> 0.2), PageRanks will oscillate and not converge; if α is too small (< 0.05), predictions are useless and the actual number of iterations will not decrease. So finally, we chose $\alpha = 0.1$.

5 Conclusion

In this project, we took the advantage of MapReduce to implement the ranks of nodes in a graph based on the transition matrix, a.k.a. PageRank algorithm. We used two sequential MapReduce process: PageRank step calculates and update the page ranks of each node; Process step determine the convergence condition and output the sorted top 20 nodes if it converges.

After implementing basic MapReduce algorithm, we found out that the whole MapReduce process is using a large space-compexity to save running time. However, due to the overhead and postprocess issues of Amazon Web Seivice System, each iteration would take at least 6 - 7 minutes. So, in term of the competition, it is essential to make iteration number as small as possible.

In the optimization process, we carefully optimized convergence condition using "top-k convergence condition", following the tradeoff between accuracy and running time. Also, we implemented heapsort to reduce the time complexity of single iteration. More importantly, we studied the pattern of PageRank of top nodes in different iterations, and proposed our core algoritm - "linear extrapolation" strategy, which helps us to predict future PageRanks and skip iterations.

As a result, our team ranked second on the scoreboard among 20 teams, and beated the champion of 2017. For future improvement, we would consider partitioning graph into subgraphs and iterate MapReduce on clusters rather than single nodes.

References

[1] PageRank - Wikipedia,
`https://en.wikipedia.org/wiki/PageRank`

[2] MapReduce - Wikipedia,
`https://en.wikipedia.org/wiki/MapReduce`

[3] Stanford Large Network Dataset Collection,
`http://snap.stanford.edu/data/index.html`