Name : Sushil Thasale
Project Report : Hadoop MapReduce Performance

My approach to implement page rank using map reduce is slightly different from the pseudo-code mentioned in course modules. The pseudo-code for my approach is as follows:

1. Preprocessor-1 Job:

// translates input text into graph using the provided parser
Map(Text):
  Process input text
  Find page P and its outlinks from processed input Text
  Increment job-1 specific counter for counting total legal pages
  PageData = [0, outlinks] //set page rank to 0
  emit(page, PageData)

Reduce(Page, List<PageData>):
  // just emit
---
2. Preprocessor-2 Job: Computes initial page rank

// total no. of pages are passed to this job

Map(Page, PageData):
  emit(Page, PageData)

Reducer:

Cleanup():
  // computes sum of page ranks of sink nodes
  set sinkSum = 0

  // compute initial page rank
  inital_page_rank = 1/total_pages

Recuce(Page, List<PageData>):
  // here List will contain only 1 PageData object
  fetch PageData PD from List
  PD.pageRank = inital_page_rank

  if Page is a sink node:
    sinkSum += PD.pageRank

  emit(Page, PD)

Setup():
  Increment job-2 specific counter for counting total sink page rank

3. Page-Rank Job: // called 10 times

/*
→ This job reads Page and its corresponding page rank computed in previous iteration. It refines page rank and writes it to sequence file.
→ Total pages is passed to this job.
→ Sink page rank computed in previous job is also passed.
*/

Map(Page P, PageData PD)
```
  // page rank is set to zero
  // this is emiited to fetch Page's outlinks in reducer
  emit(Page, new PageData(0. PD.outlinks))

  for each outlink O in PD.outlinks:
    // compute P's contribution towards page rank of O
    PageData out = new PageData()
    out.pageRank = P.pageRank / length(PD.outlinks)
    out.outlinks = ""
    emit(O, out)
```

Reduce(Page P, List<PageData>):
```
  newPD = new PageData()
  pageR = 0

  for each PD in List:
    pageR += PD.pageRank
    if PD.outLinks is not null:
      newPD.outlinks = PD.outlinks

  // add teleporation component to pageR
  // add sink sum component to pageR
  newPD.pageRank = pageR
  emit(Page P, newPD)

  // also compute new sink sum as specified in previous job
```
---
4. Top-100 Job:

Map(Page P, PageData PD):
```
  // this done to take advantage of built-in sorting
  emit(PD.pageRank, Page P)
```

// custom key comparator is used to sort keys in descending order

Reduce(pageRank, List<Page>):
```
  //emit only top-100 values by maintaining a counter
  for(P: List):
```

```
    if count < 100:
        emit(P, pageRank)
```

Analysis over data transferred (local execution):

	Data Transferred		
Job	**Map Output Bytes**	**Reduce Input Records**	**Bytes Written**
Preprocessing-1	4684786	18328	4901648
Preprocessing-2	4684786	18328	4901648
Page Rank Iteration-1	12674624	433190	4904747
Page Rank Iteration-2	12677087	433264	4904747
Page Rank Iteration-3	12677087	433264	4904747
Page Rank Iteration-4	12677087	433264	4904747
Page Rank Iteration-5	12677087	433264	4904747
Page Rank Iteration-6	12677087	433264	4904747
Page Rank Iteration-7	12677087	433264	4904747
Page Rank Iteration-8	12677087	433264	4904747
Page Rank Iteration-9	12677087	433264	4904747
Page Rank Iteration-10	12677087	433264	4904747
Top-100	405646	18402	3030

In all page rank iterations, the amount of data transferred from mapper to reducer is 12674624 bytes. The number of input records to reducer and bytes written to HDFS, also remain constant. In mapper of page-rank job, for each of the outlinks of a given page we make one emission. Since, the graph remains constant overtime, every node will have same number of outlinks over 10 iterations. Hence, the data transferred between mapper to reducer remains constant over page rank jobs.

---

1. Time take for executing Page-Rank on six M4.large machines:

→ Time for preprocessing = 273 seconds
→ Time for 10 page rank jobs = 1217 seconds
→ Time for top-100 job = 59 seconds

2. Time taken for executing Page-Rank on eleven M4.large machines:

→ Time for preprocessing = 76 seconds
→ Time for 10 page rank jobs = 808 seconds
→ Time for top-100 job = 40 seconds

All page-rank iterations and top-100 job showed a good speedup.
→ Speedup for preprocessing = 3.5921
→ speedup for entire page-rank job = 1.506
→ speedup of top-100 job = 1.475

Preprocessing phase with 11 machines showed best speedup of 3.5921. Top-100 job showed least speedup. This is because we are using a single reducer in Top-100 job to emit top 100 pages. So there is no scope of speeding the reducer computation using 10 worker machines.

Top-100 Wikipedia pages with highest pages have been reported under **“All output and log files”** folder.

The top pages include “United_States”, “England”, “Europe”, “Germany”, “Asia”, “United Kingdom” etc. Based on my intuition, these values seem reasonable. Because many things might be related to these pages, and so these pages might have large number of inlinks that contribute towards its page-rank. As a result, page-rank of these page is likely to be high.
