CS 6240: Assignment 5

Xiao Wang
GitHub: https://github.ccs.neu.edu/xiaowang/CS6240-MapReduce

1. Design Discussion

Version A:

PreProcessor

```
parse(line):
    // removes self-loops as they affect PageRank results
    returns [pageName, [link]]
map(line):
    String[] s = parse(line)
    emit(s[0], s[1])
reduce(page, links):
    pageIndex[page] = nPage++
    nOutlinks[page] = len(links)
    for link in links:
        inlinks[link].append(page)
cleanup():
    // handle dangling nodes
    for page if nOutlinks[page] == 0:
        inlinks[all pages].append(page)
    writeMapping(pageID:pageName)
    writeMatrix(pageID:inlinkID,v|inlinkID,v|...)
    writeInitPR(1.0 / nPage)
```

Iterator

```
setup():
    lastPR = read(lastIterationOutput)
map(pageID, row): // row in M
    score = 0
    for j, v in row:
        score += v * lastPR[j]
    score = score * (1 - alpha) + alpha / nPage
    emit(pageID:score)
```

Ranker

```
map(page, score):
    emit(score, pageID)
partition(score, pageID):
    return 0
compare(w1, w2):
    return -w1.compareTo(w2)
setup():
    pageIndex = read(mapping)
reduce(score, pageID):
```

```
    while (count < 100):
        emit(pageIndex[page], score)
        ++count
```

Version B:

PreProcessor

```
parse(line):
    // removes self-loops as they affect PageRank results
    returns [pageName, [link]]
map(line):
    String[] s = parse(line)
    emit(s[0], s[1])
reduce(page, links):
    pageIndex[page] = nPage++
    nOutlinks[page] = len(links)
    for link in links:
        outlinks[page].append(link)
cleanup():
    // handle dangling nodes
    for page if nOutlinks[page] == 0:
        outlinks[page].append([all pages])
    writeMapping(pageID:pageName)
    writeMatrix(pageID:outlinkID,v|outlinkID,v|...)
    writeInitPR(1.0 / nPage)
```

Iterator

```
setup():
    lastPR = read(lastIterationOutput)
map(pageID, col): // col in M
    score = 0
    for i, v in row:
        emit(i, v * lastPR[pageID])
reduce(pageID, [score]):
    total = sum([score])
    emit(pageID:total)
```

Ranker

Same as version A

Q: Briefly explain how each matrix and vector is stored (serialized) for versions A and B. Do not just paste in source code. Explain it in plain English, using a carefully chosen example. See the above discussion about dense and sparse matrices for an example how to do this.

A: I use sparse representation for both versions. And the serialized string format I design is: PageID:linkID,v|linkID,v|... For version A, each linkID is an inlink, thus each line represents a row in M. For version B, each linkID is an outlink, thus each line represents a column in M.

Q: Discuss how you handle dangling nodes: Do you simply replace the zero-columns in $\mathbf{M}$ and then work with the corresponding $\mathbf{M}'$, or do you deal with the dangling nodes separately? Make sure you mention if your solution for dangling nodes introduces an additional MapReduce job during each iteration.
A: I handle dangling nodes in reduce cleanup to fill the columns of all zeros.

2. Performance

Working with original inputs:

time (ms)	Version A		Version B	
	5 workers	10 workers	5 workers	10 workers
Step 1+2	91529	90741	92383	89372
1 Iter in Step 3	29260	28206	54206	42927
Step 4	31206	32213	32947	33824
Total	418476	428534	665853	544292

Q: For each configuration, report the running time of an iteration executed by the adjacency-list based Hadoop implementation from the earlier HW assignment. How do these numbers compare to the adjacency-matrix based version? Briefly discuss if you find the result surprising and explain possible reasons for it.
A: This matrix algorithm doesn't run faster than the adjacency-list version and Version B runs even slower. I think the cause is because although we use matrix representation, the computations are basically the same except we use two different files to store the graph and pagerank while the adjacency-list version uses only one which maybe more efficient.