CSCI4180 Tutorial-6

Parallel Dijkstra's Algorithm

ZHANG, Mi

mzhang@cse.cuhk.edu.hk

Nov. 5, 2015

Definition

- Model the *Twitter network* as a *directed graph*.
- Each user is represented as a *node* with a unique positive integer as the node ID.
- When user 1 follows user 2 on Twitter, an *edge* is created from node 1 to node 2 in the graph.
- An integer weight is attached to each edge, which is between 1 and 50 inclusively.

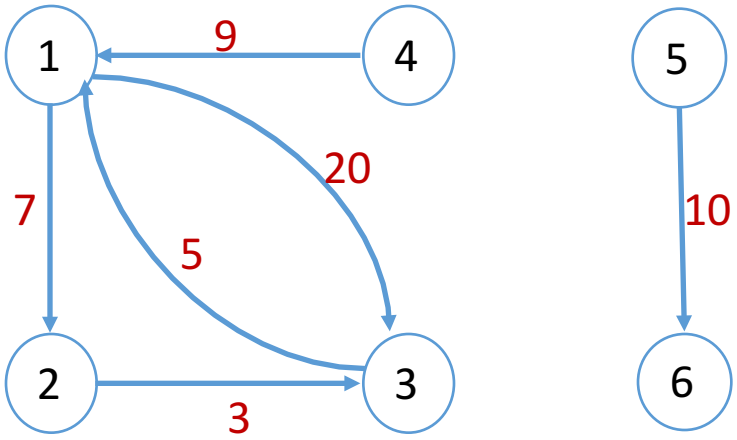
Definition

- Model the *Twitter network* as a *directed graph*
  - $G = (V, E)$
- Take twitter_dist_1.txt for example
  - 1173 1173 10
    - Add node 1173 to V
    - No edge is added to E
  - 1173 6267522 14
    - Add node 1173, 6267522 to V
    - Add edge from 1173 to 6267522 to E, whose weight is 14

Problem

- Given a graph $G = (V,E)$ and a source node $v_s \in V$, find the shortest path length (distance) from v_s to every other reachable node in V.

Input
1 2 7
1 3 20
2 3 3
3 1 5
4 1 9
5 6 10



Parsing Input

- From *Input* Format to *adjacency list* format

Input
1 2 7
1 3 20
2 3 3
3 1 5
4 1 9
5 6 10

Adjacency List
1 (2 7), (3 20)
2 (3 3)
3 (1 5)
4 (1 9)
5 (6 10)

- Hint:
  - You could design a separate Mapper/Reducer to do the transformation

Node Structure

- Stores distance from v_s as internal state
  - Here distance means the shortest path length
- Keeps track of neighbors by adjacency list
  - Adjacency matrix is difficult for MapReduce

Design Node Class

- Choose proper type for node ID and distance
- Choose proper type for the adjacency list
- Design a proper constructor

- Think about the initial `distance`
- Feel free to design your own format

Emit a Node

The Java Way

- Implements a toString() method and a fromString() method in your Node Class
- Emit a node simply by a org.apache.hadoop.io.Text
- However, functions like `String.split()` could be slow

Emit a Node

The Hadoop Way (Recommended)

- Implement *Writable* in the Node Class
  - ➢ You will need to provide implementation for the readFields() and write() method
- Emit a Node as the value directly
- Could be faster than parsing Strings in the Java way, worth trying if you want to go for Bonus!

Parallel Dijkstra's Algorithm

- Mapper
  - For each node
    - calculate neighbors' distance
    - emit itself and its neighbors
- Reducer
  - For a Node, different mappers will provide conflicting information
  - Think about how to update a node's distance using the smallest value
  - The emitted node can then be used for the Mapper in the next iteration

Chain MapReduce Jobs

- Each "Map + Reduce" only explored one step further from v_s, we need to repeat this process
  - Map1, Reduce1, Map2, Reduce2, Map3...
- How to chain MapReduce jobs?

Chain MapReduce Jobs

The Java Way

- Drive jobs in the main() function using loops, counter and conditionals
- Different Configuration and Job Object in each iteration
- Set job.waitForCompletion(true);
- Jobs communicates by writing and reading intermediate files on HDFS
- For Job_i :

```
FileInputFormat.addInputPath (OutputPath of Job_{i-1})
```

Chain MapReduce Jobs

The Hadoop Way

- Use org.apache.hadoop.mapred.jobcontrol
- Add jobs to JobControl
  - JobControl jc = new JobControl();
  - jc.addJob(job1);
  - jc.addJob(job2);
- Add dependency (e.g. 2 depends on 1)
  - job2.addDependingJob(job1)
- Run JobControl
  - jc.run()

Stop Condition

- *Iterations*
  - Command-line argument
  - To indicate the number of mapreduce iterations
  - Non-negative integer
- Two conditions:
  - *Iterations* > 0
    - Stop after *iterations* runs of mapreduce
  - *Iterations* = 0
    - Stop when all reachable nodes are processed
    - How to check?
      - We could use org.apache.hadoop.mapred.Counters

Hadoop Counter

- Declare Counter

```
public static enum ReachCounter {
        COUNT
};
```

- Increment Counter

```
context.getCounter(ReachCounter.COUNT).increment(1);
```

- Retrieve Counter Value

```
long reachCount =
job.getCounters().findCounter(ParallelDijkstra.ReachCounter.COUNT).getValue();
```

Clean-up

- Transform the output of the last run to the required format
- Only outputs the tuple which the node is reachable
- Feel free to use another set of Mapper/Reducer to do this step

Tips

- Test your program correctness with hand-craft test cases (loops, directed edge, etc.)
- Should be more challenging than Assignment 1
- Please start early!

Thank you!