SDM Lab2: Pregel

Yi Wu
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
yi.wu@estudiantat.upc.edu

Hang Yu
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
hang.yu1@estudiantat.upc.edu

1 Exercise 1

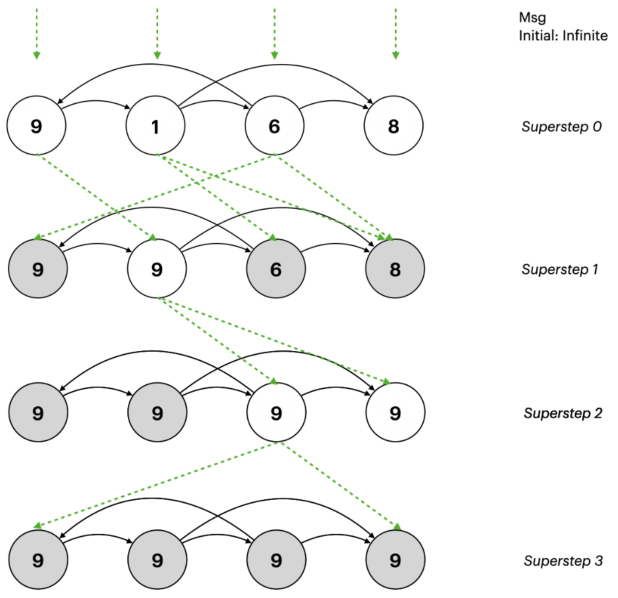

Figure 1: Maximum Value Supersteps

In superstep 0, the initial message is set as an Int.Max_Value. All nodes get those messages, then process those messages via VProg, and update their states as activated.

In superstep 1, all activated nodes temp to send messages to their neighbors via sendMsg. Function merge is used to aggregate and take the max incoming message. Function VProg is used to compare the incoming message with the vertex value. If the source-node value is higher than the destination-node value, then the destination-node will be triggered, and update their status as activated. In this case, the 2nd node becomes activated, other nodes vote to halt.

In superstep 2, the 2nd node will keep sending messages to its neighbors. The 3rd and 4th nodes change their values, activated by the messages sent by the 2nd node from the last super-step.

In superstep 3, the 3rd node tries to send messages to the 1st and the 4th nodes but does not succeed. All nodes are deactivated. The loop ends.

2 Exercise 2

Two ArrayLists are used to store the values of vertices and edges. We initialized the values of vertices with the Int.Max_Value except the vertex A. The value of vertex A is set as 0, which means to compute the shortest path from A to all other ones. Then ArrayLists are injected into RDD as a component to form the graph. After initialization, GraphX's Pregel-style vertex-centric programming model is applied.

In superstep 0, the initial message is set as an Int.Max_Value. All nodes get those messages, then process those messages via VProg, and update their states as activated.

In superstep 1, Vertex B receives one message with a sum value of Vertex A 0 and edge value 4. Since it's smaller than Vertex B's own value Int.Max_Value, Vertex B updated its value as 4, and become activated. So does vertex C, with updated value 2. Vertex A get no messages. Other get messages with value Int.Max_Value, no smaller than their own value Int.Max_Value, so remain inactive.

Then in superstep 2, Vertex B sends two messages to Vertex C and D. In a similar way, the value of Vertex C is still 2 after comparing the values of 2(from Vertex A) and (4+5=9 from Vertex B). The value of Vertex D became 14(4+10) after comparing with Int.Max_Value. Vertex E receives the message from Vertex C and the its value becomes 5(2+3). The values of Vertex B and C does not change, so they became inactive. Only vertex D and E are activated.

In superstep 3, D receives a message from E and its value changes to 9(2+3+4<4+10). Vertex F's value became 25(4+10+11). Vertex E became inactive. Only vertex D and F remain activated.

In superstep 4, only Vertex D sends message to Vertex F and Vertex F's value updates to 20(2+3+4+11).

In superstep 5, all vertice's values remain same. They all become inactive.

Finally, each final vertex value represents vertex's distance to vertex A. The minimum cost from Vertex A to other vertices is generated.

3 Exercise 3

In exercise 3, the vertex is created with parameters of a value to store the shortest path length, and a list to store a set of vertices that the path contains. The supersteps iterate like Exercise 2, and in each step, the path is stored in the vertex list.

4 Exercise 4

Iteration times is set from 4, 6, 8,..., 20. Damping factor is set from 0.9, 0.8,..., 0.6. As shown in Figure.2, the x-axis is the damping factor, the y-axis is the time consuming. And the iteration times is displayed in different colors of lines, from bottom to top, indicated as iteration times from 4 to 20.

We first tried damping factor from 0.9, 0.8,...,0.4, and found that 0.8 has perform the best results. According to typical experience, optimal damping factor shall be at 0.85. So we further tested 0.825, 0.85, 0.875 as well.

The experimental results has shown that, damping factor between 0.8 to 0.825 has given the best performance, and an iteration of 4 has achieved the fastest speed. The corresponding 10 most relevant papers are as follows.

```
Damping Factor: 0.8
Reset Prob: 0.2
Max Iterations: 4
Time(ms): 43221
22/04/25 17:09:22 WARN TaskSetManager: Stage 8446 contains a task of very large
    size (174 KB). The maximum recommended task size is 100 KB.
```

id	name	pagerank
|8830299306937918434	University of Cal...	3020.557522415592
|1746517089350976281	Berkeley, California	1491.365137941174
|8262690695090170653	Uc berkeley	367.10811415488416
|7097126743572404313	Berkeley Software...	199.2262354257445
|8494280508059481751	Lawrence Berkeley...	184.30363343083965
|1735121673437871410	George Berkeley	181.8036918760219
|6990487747244935452	Busby Berkeley	99.92462859479932
|1164897641584173425	Berkeley Hills	94.61887124467253
|5820259228361337957	Xander Berkeley	66.41022513826593
|1630393703040852365	Berkeley, CA	61.92961417815851

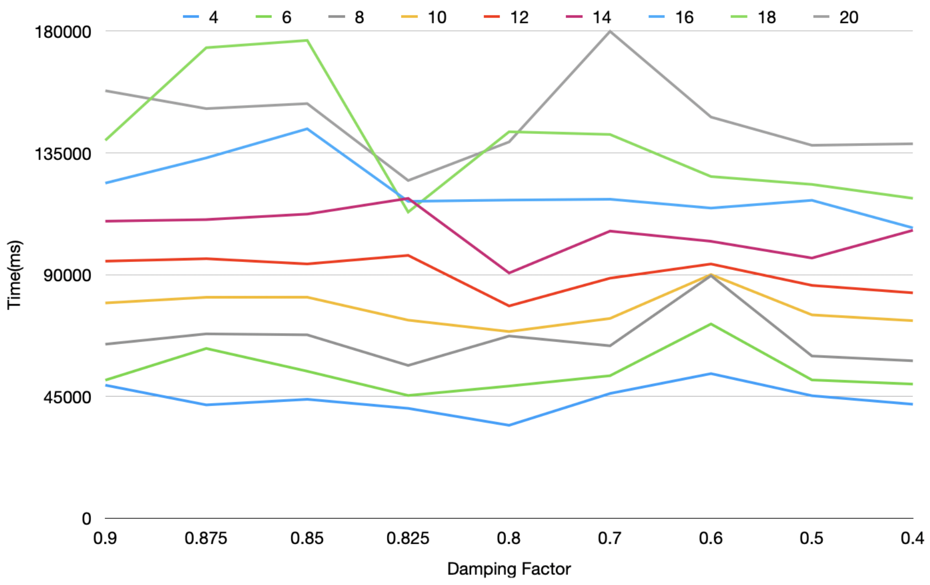

Figure 2: Experimental Results of Various Iterations