Universitat Politècnica de Catalunya

Semantic Data Management

LAB Assignment 2

Distributed Graphs

Enric Reverter
enric.reverter@estudiantat.upc.edu

Dani Muñoz
daniel.munoz.checa@estudiantat.upc.edu

Curs 2022/2023 Q1

Contents

A Distributed Graphs

A.1 Exercise 1: Getting familiar with GraphX's Pregel API

In this exercise, the TLAV methodology is explored using the Pregel framework to compute the maximum value for the graph depicted in Figure 1. It is worth noting that the supersteps are illustrated within the same figure.

Given the provided Java implementation, the algorithm converges as described below. It is important to mention that in this example, the merge (*Gather*) substep does not play a role since each node receives at most one message at a time. If a single vertex were to receive more than one value, the function should be modified to return `Math.max(o, o2)` instead of the current `null`.

Superstep 0:

- merge (*Gather*): nothing to merge.
- VProg (*Apply*): each node receives an initial message with a large value ($\rightarrow \infty$) to initialize their state with their values.
- sendMsg (*Scatter*): the first messages are sent. Since triplet views are available to determine which information needs to be sent, only vertex 1 (9) propagates its value to its neighbors. Specifically, the values for pairs 1-2, 2-3, 2-4, 3-1, and 3-4 are compared, but only the message from 1 to 2 is sent.

Superstep 1:

- merge (*Gather*): nothing to merge.
- VProg (*Apply*): node 2 computes the maximum of its received messages (9) and its current value (1) and sets its new value as 9. The rest of the nodes have not received any messages, so they are inactive.
- sendMsg (*Scatter*): node 2 compares its current value (9) to its neighbors, node 3 (6) and node 4 (8), which both have lower values, so it sends its value to them.

Superstep 2:

- merge (*Gather*): nothing to merge.
- VProg (*Apply*): nodes 3 and 4 have received a message from node 2, and they set their new value to the maximum of the received messages and their current values.
- sendMsg (*Scatter*): since nodes 3 and 4 have been updated, the triplet views of their neighbors are checked to determine if they need to propagate their values. Since all values are now 9, no more messages need to be sent. Consequently, convergence is achieved, and the termination criteria are met according to the Pregel framework.

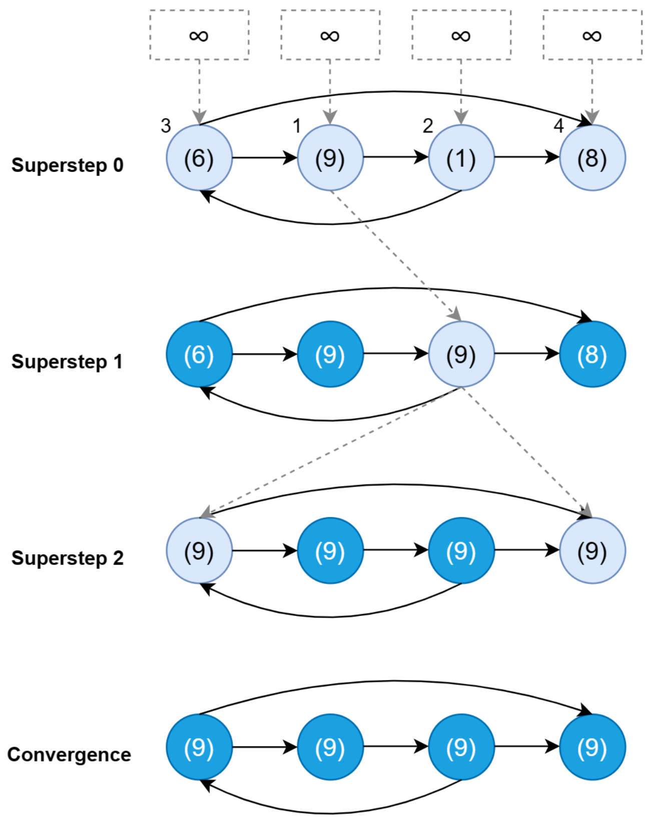

Figure 1: Supersteps during the Pregel implementation.

A.2 Exercise 2: Computing Shortest Paths Using Pregel

In this exercise, the Pregel framework is used to compute the shortest path from an initial node to all others. Thus, the same GAS approach is considered. The following assumptions regarding the network are considered:

- The graph is assumed to have non-negative edge weights. Negative edge weights can lead to incorrect results, as the algorithm does not account for negative cycles.
- The implementation assumes that the graph is connected, meaning that there is a path between every pair of vertices.

Then, the GAS approach is defined as follows:

- merge (*Gather*): combines messages sent to a vertex by taking the message with the shorter distance.
- VProg (*Apply*): updates the vertex value (path cost) by taking the minimum distance between the current vertex value and the received message. In the first superstep (0), vertex values are initialized with their initial values. In subsequent supersteps, the vertex values are updated with the minimum distance.
- sendMsg (*Scatter*): sends a message from the source vertex to the destination vertex if the source vertex distance plus the edge weight is less than the destination vertex distance. This is possible thanks to the use of triplet views.

A.3 Exercise 3: Extending Shortest Path's Computation

In this exercise, the aim is the same as the one in Exercise A.2, but storing the path as the algorithm converges. The assumptions are the same, and the GAS approach is almost equivalent. The only difference is that vertices are now defined as `Tuple2<Integer, List<Long>>` instead of `Integer`. Hence, when messages are sent, the list containing the current path is also carried. The path is updated during the *Scatter* substep if the message is sent. A helper function is also defined to print the cost and path after convergence.

A.4 Exercise 4: Spark Graph Frames

The aim of this section is to show the potential that Spark GraphFrames can offer. This Apache Spark package provides DataFrame-based Graphs for Scala, Python and Java, which is the language that has been used in this project. These new data structures posses the functionality of GraphX seen in the previous exercises and further functionalities implemented in the package.

For this exercises, a dataset that contains 31312 Wikipedia articles and 22424 relations between them is used.

Spark Graph Frames present a wide collection of algorithms suitable for the TLAV methodology such as: **Search-first search**, **Label Propagation Algorithm**, **PageRank** and **Triangle count** among others.

This exercise consists in finding the top 10 most relevant articles in the dataset. To do so, the PageRank algorithm, included in the package, is applied. Two parameters need to be defined: the reset probability, which has been defined with one of its most common values (0.85), and the number of maximum iterations, which is set to 10 as it is already enough to converge.

The final top 10 most relevant articles are:

Article	**PageRank score**
University of California, Berkeley	3119.05
Berkeley, California	1574.63
Uc berkeley	384.33
Berkeley Software Distribution	214.48
Lawrence Berkeley National Laboratory	194.84
George Berkeley	193.04
Busby Berkeley	110.93
Berkeley Hills	108.01
Xander Berkeley	70.85
Berkeley County, South Carolina	67.72

Table 1: PageRank results.