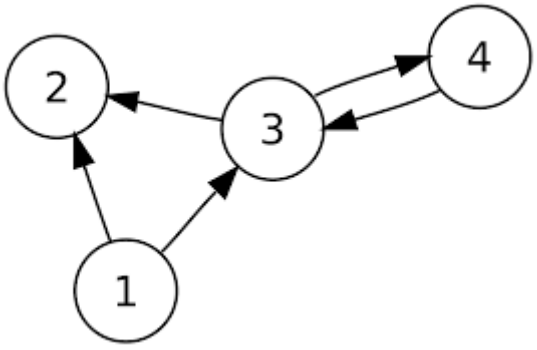
Atividade Dirigida

CSC-27
DCTA - ITA - IEC
Gabriel Gandour

Goal: To present the Pregel framework, as well as its alternative Go-Pregel for distributed graph processing. Furthermore, it is also a goal to execute a hands-on activity to convert a sequential centralized graph algorithm to a distributed version.

Background

Pregel is a Google framework used for writing distributed graph algorithms.



Go-Pregel is a Golang version of Pregel, written by Gandour for his Graduation Thesis. Its basic functionality is the same as the original Pregel, and both frameworks work in the same way (although Google's Pregel is written in C++). From now on, for simplicity, all mentions of Pregel refer to Go-Pregel.

The Problem

The objective of Pregel is to distribute graph algorithms. Imagine Facebook's social network as a graph, where each vertex is a person and each edge is a friendship between two people. Or imagine the internet as a graph, where each vertex is a website and each edge is a hyperlink. These are huge graphs in which you can't simply process all the data in a single machine, let alone run a graph algorithm on

it, such as Dijkstra's algorithm or the PageRank algorithm. We thus need to distribute the computation.

However, it is difficult to distribute the computation of a graph algorithm, because each algorithm has its own characteristics and can be parallelized in a different way. Pregel solves this problem by providing a framework that allows the user to write a graph algorithm in a simple way, and then Pregel takes care of distributing the computation. Of course, it isn't as simple as copying the algorithm to Pregel, but the necessary adaptations are much simpler that writing a distributed algorithm from scratch.

Input and Output

The input of a Pregel algorithm is always a graph, and the output of a Pregel algorithm is always a graph (there are some exceptions that won't be covered in this document). However, in some algorithms, the desired output is not a graph. For example, in the SSSP problem, the output is maybe a path from a source to the destiny, along with the distance traveled.

We can't make the Pregel algorithm print the path, nor the distance, but we can store the minimum distance in each vertex, and we can store the neighbor vertex that leads to the shortest path to the origin.

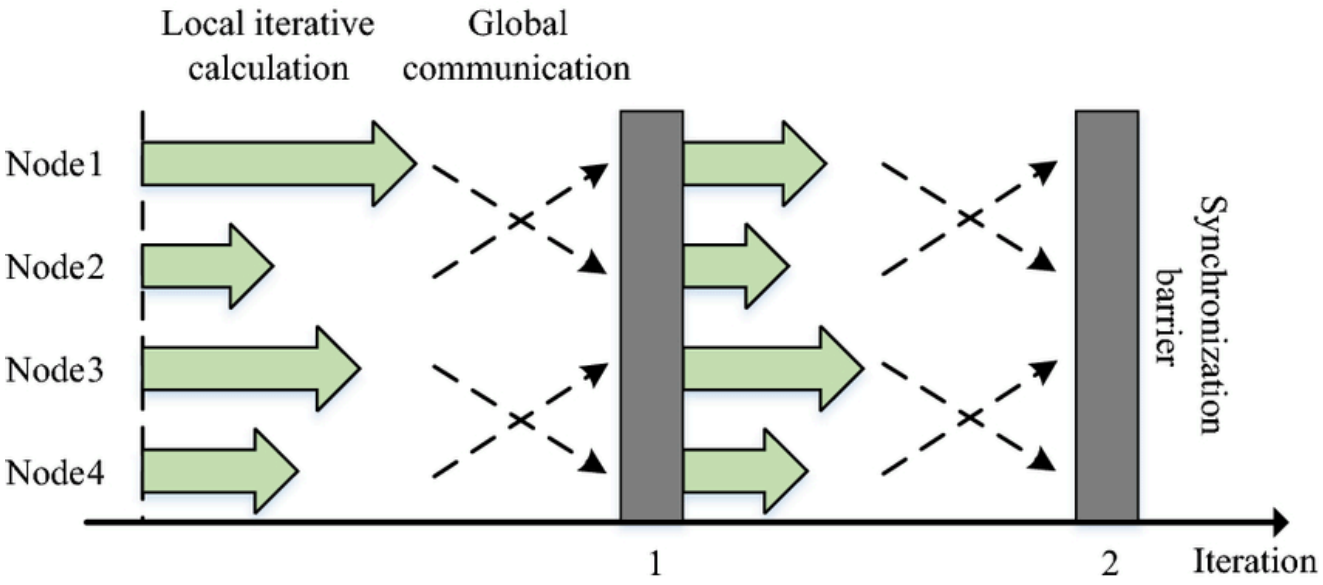
The key idea is to always save important information inside each vertex of the graph, even in the middle of the algorithm.

Bulk Synchronous Parallel (BSP)

Pregel uses the BSP paradigm to execute the algorithm, together with a master-worker architecture.

In BSP, the algorithm is divided into several blocks named Supersteps. In each superstep, each worker (or node) does a certain amount of local computation, followed by communicating the necessary information with the other workers. After communication, each node meets a synchronization barrier, which stops them from proceeding to the next superstep until every other worker arrives at the same barrier. When every node is at the barrier, they are allowed to go to the next superstep.

In Pregel, the Synchronization barrier is managed by the master machine. The main responsibility of the master machine in Pregel is telling the workers when they can proceed to the next superstep.



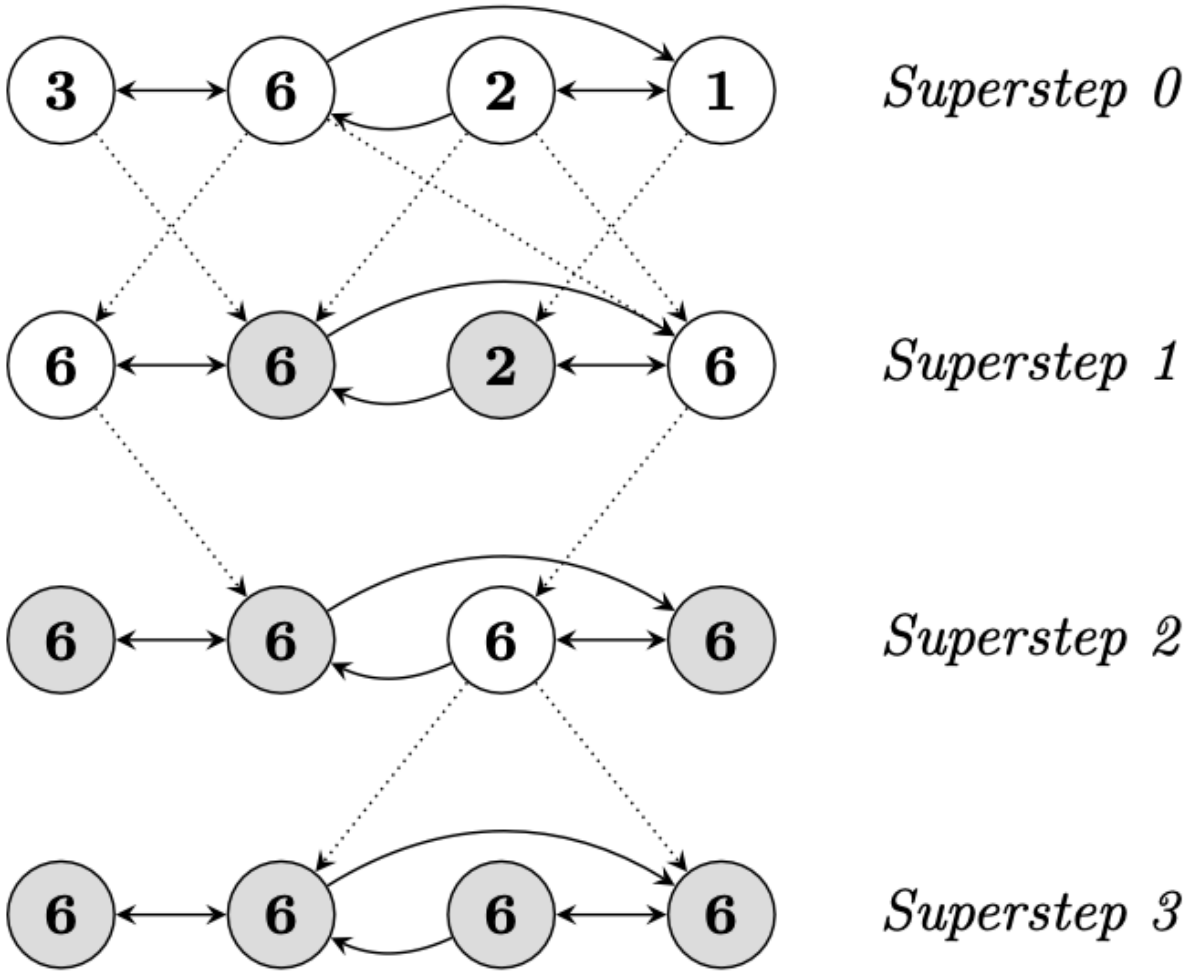
Pregel and BSP

In Pregel, each vertex of the graph acts as an independent node, with its own computation. Each vertex is capable of changing its own internal state, sending messages to other vertices, receiving messages and interpreting them.

BSP shows us how Pregel executes its computation, but it doesn't show us how the algorithm stops. The algorithm stops by unanimity. After each superstep, each vertex votes if Pregel should or not halt. If every vertex agrees to halt the algorithm, it stops, and the final graph is written. When a vertex votes to halt, it becomes inactive, and it won't take part in the next computation steps. The vertex stays inactive forever, or until it receives a message from another vertex (after which it becomes active again, and the vote to halt is nullified until it votes again).

The best way to understand it is by watching a real Pregel execution. The next image shows the execution of a Pregel algorithm. The input and output are strongly connected graphs in which each vertex has an integer value. The algorithm finds the minimum value among all the vertices and stores it in every vertex. The white vertices are active and the gray ones are inactive. The full arrows are edges and the dotted arrows are messages being passed. Finally, the number inside each vertex is the value it holds in a certain

superstep. We'll call the vertex in the image below, from left to right, as A, B, C and D.



In the superstep zero, vertex A sends a message to B saying that A has value 3. On the other hand, B sends a message to A and D saying that B has 6. In superstep 1, the vertexes read the messages received in superstep zero and update their values - A and D change to 6, while B and C didn't change their values. Since B and C didn't change values, they vote to halt. The active vertices then sent messages to the others: A sends 6 to B and D sends 6 to C. In superstep 2, C receives a message, so it awakens and changes value. B also awakens, but it doesn't change value and votes to halt again. Finally, C sends 6 to B and D. In superstep 3, B and D receive the message, but don't change value. They vote to halt, and the algorithm stops because every vertex is inactive.

Algorithm Distribution

Note that each vertex is independent, and only communicate with others through messages. There is no memory sharing between them. So

it's possible to distribute the vertexes among many machines. In practice, each machine receives many vertexes, and the whole graph is distributed uniformly among those machines.

Activity

1. Gather in pairs. If you want, you can also do the activity alone. However, even if you do it in pairs, each person has to write the code and make it work on their own computer.
2. Go to https://github.com/GaGandour/Go-Pregel and clone the repo.
3. Read the README.md to set up your environment. All the commands have to be run from the project directory.
4. Read the pregel_writing_guide.md file to understand which files you have to modify, if necessary.
5. Start coding!

There will be three levels of difficulty, to be done in order. For each level of difficulty, you have to:

1. Write the algorithm in Go-Pregel.
2. Test the algorithm using the provided testing script (read the repo README.md)
3. Be sure that you algorithm passes all tests.

The Levels

Level 0 (Very Easy)

Implement the Single Source Shortest Path (SSSP) algorithm in Go-Pregel, and then test the algorithm on the given graphs. The solution is in the pregel_writing_guide.md file.

Level 1 (Easy)

Choose between one of the following algorithms:

1. Connected Components
2. Topological Sort

Level 2 (Medium)

Implement the Bipartite Graph algorithm in Go-Pregel, and then test the algorithm on the given graphs.

Level 3 (Ultra hard - Optional)

Solve Sudoku using Pregel. You have to be creative here.

Deliverables

1. For each level of difficulty, the files `./src/graph_package/user_defined_graph_types.go` and `./src/graph_package/user_defined_graph_methods.go` must be submitted.
2. In the `./src/graph_package/user_defined_graph_methods.go` file, write a comment explaining how you came up with the algorithm adaptation to Pregel. Don't forget to put your name and the name of the algorithm on the top of the file.
3. Fill in the google form with the link available in Google Classroom.