Pregel: Distributed System for Large Graph

Qiyuan Qiu
School of Arts and Sciences
Computer Science
University of Rochester

***Abstract*—Large graph is ubiquitous. Examples include Facebook's relationship graph, Google's PageRank algorithm. The scale of these graphs are the main challenge. They are usually billions of vertices and trillions of edges. Targeting at this problem, this paper discusses an abstraction of large scale graph programming – Pregel. It focuses on vertices in the graph. Programs are executed in iterations. During each iteration, each vertex in the graph can do the following things: receive message from the previous iteration; send message to other vertices; modify its own state and the state of its outgoing edges; change graph topology.**

I. Introduction

Graph is a common model for various practical issues. Typical examples are "similarity of newspaper articles, paths of disease outbreaks, or citation relationship among published scientific work". Popular algorithms on graphs shortest path, clustering and PageRank, minimum cut etc.

Due to the poor locality of memory access, very little work per vertex and changing degree of parallelism over course of execution, processing large graph is hard. The existing approaches to run an algorithm on a large graph is typically the following:

1) Setting up a customized hardware environment specifically for the algorithm that is going to run on the given graph
2) Using existing programming abstraction for large graphs like MapReduce. However, MapReduce was not designed for this particular algorithm. To make the algorithm work using MapReduce often requires a lot of tunning.
3) Choosing from some single machine algorithm library for the algorithm. This approach usually has scalability issues.
4) Choosing from some existing parallel graph systems. This approach does not have a mechanism for fault tolerance.

All of these are not ideal way to program algorithms on large scale graphs. However Pregel is the abstraction that is highlighting fault-tolerance and scalable to address this challenge.

Pregel computations is based on global iterations or *superstep*. Synchronizations happen within superstep. Iterations continue until termination conditions are met. For each superstep, the Pregel calls a user defined function attached to each vertex. The function can do the following things: It can read message from the previous superstep; send message to the next superstep; modify the state of itself and its outgoing edges. Messages can be sent to any vertex (if that it receives the identifier of other vertices)

This abstraction is similar to MapReduce in that clients focus on local computations(on vertices) work on their computations independently, and the server combines these results to a larger dataset. Because the computation is independent with each superstep and all communications happen between supersteps, this abstraction is easier to analyze and implementation is free from deadlocks and data races issues exist in most other common asynchronous systems.

II. Model of Computation

The input to the Pregel abstraction is a directed graph. In that graph, each vertex is given a *vertex identifiler*. It is used to refer to each vertex. Each vertex has a value that could be modified by user associated with it. Edges in this model is attached to vertices. It could also be assigned values.

While execution, Pregel takes in a graph and keep running from superstep to superstep until terminate conditions are met. At that point it outputs a resulting directed graph.

Within each superstep, the vertices compute the same user defined function. The algorithm run on the large graph is encoded in this user defined function. All these vertices run in parallel.

As is described before, each vertex can do the following things: 1. modify its own state. 2. modify its outgoing edges' states. 3. receive message sent to it from the previous superstep. 4. send messages to other vertices whose identifiers are known to this vertex. 5. mutate the topology of the graph (can remove edges and itself). Edges in the graph dose not have associated user defined function.

The termination of Pregel depends on two conditions. The first is if there exist any vertex that is active. If there is still active vertex, Pregel will not terminate. The second condition is if there are any messages unsend. If there is message has not been sent, the abstraction will not terminate. A graph from the original Pregel paper [**?**] illustrate the state machine for the active and inactive state transition for each vertex.

A. Subsection Heading Here

Subsection text here.

1) Subsubsection Heading Here: Subsubsection text here.

III. Conclusion

The conclusion goes here.

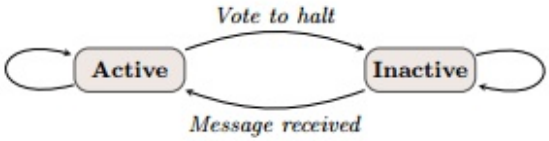

Fig. 1. Vertex State Machine

ACKNOWLEDGMENT

The authors would like to thank...