Simulation of Streaming Graph Partitioning

Yangguang Liao, Siqi Wu, Liwei Wu
UC Davis
November 2015

1 Introduction

Since today's data set scale is huge, large-scale graph-structured computation is critical for most modern computation problem, which has led to the development of graph-parallel abstractions. There are huge amount of graph based data, for example the large web search engine Google crawls over about ten trillion links of world wide webpage. User relation in Facebook is also a typical graph structured data set with billions of friends link. Other social media network also provide valuable graph structured data, In July 2009, Twitter had over 41.7 million users with over 1.47 billion social relations. Examples of large graph datasets are not limited to the Internet and social networks, biological networks, like protein interaction networks, are of a similar size. Astronomy and planetology science has same size scale or larger astrophysical data. People characterize the essential issue of graph-based parallelism by clustering and graph partition to maximize the computation performance and penalty for cluster communication.

Typical method for dealing with large scale graph data is using reasonable partition algorithm to split the data across a large cluster of commodity machines and use parallel, distributed algorithms for the computation. This approach introduces a host of systems engineering problems of which we focus only on the problem of data layout. For graph data, this is called balanced graph partitioning. The goal is to minimize the number of cross partition edges, while keeping the number of nodes (or edges) in every partition approximately even.

There are various reasons for us to apply a good graph partition algorithm. First, graphs that we encounter and care about in practice are not random. The graph definitely reveal some nature of our real world. The edges display a great deal of locality, whether due to the vertices being geographically close in social networks, or related by topic or domain on the web. This locality gives us hope that good partitions, or at least partitions that are significantly better than random cuts, exist in real graphs. Second, inter-machine communication, even on the same local network, is substantially more expensive than inter-processor communication. Network latency is measured in microseconds while inter-process communication is measured in nanoseconds. This disparity substantially slows down processing when the network must be used. For large graphs, the data to be moved may change the communication structure, causing network links to become saturated.

With the scaling up of the system, big graph has to be cut into several pieces and then loaded into several clusters. Existing graph partitioning algorithm usually have very large cost, may be as large as future computation cost. However, since the graph need to be loaded into clusters anyway, we simulate the streaming graph partitioning, i.e. partitioning the graph at the same when the graph is loading into clusters.

2 Related work

There are a lot of related research in this topic. The most famous one, which is regarded as NP-Hard, is to cut a graph into k balanced parts, while minimize the number of edges being cut. Stanton and Kliot discuss several partition methods, and streaming orders of the graph, and their model is proved better than the partitioning method used in Spark.

3 Methodology 1 heuiristic 1-9 chuiable

2 streaming order

implementation why python or library usage datastructure dfs bfs algorithm

The algorithm is intuitive, vertices are coming in a stream, and several different methods were provided, like Balanced, Randomized Greedy, greedy EvoCut, to determine where should we put the new node. The streaming order like BFS, DFS or random is also considered here. In this way, we could partition the graph into several pieces when loading it into clusters. We simulate the combinations of dataset, streaming order and heuristics on our own offline system to evaluate their performance.

Different streaming strategy and ordering could fit different dataset type. Our experiment is trying to find the most reasonable heuristic for specific graph type. To find out relationship between computation performance and scaling size, data features and graph properties.

Experiment dataset description simulation pre processing format transformation streaming simulation

Dataset type
The datasets are chosen from the paper and online data library SNAP. The data size would be small? since we are expected to using offline system to simulate the heuristics implementation. For each dataset, we will record its scaling size, data feature and graph properties.
Evaluation method
We will simulate each heuristic with 3 different ordering strategies on all the datasets 5 times, which could reveal an average performance for each combination.
Quality evaluation
We use following approach to evaluate heuristics quality:
1. Upper bound: RANDOM HASHING and lower bound METIS.
2. Time cost improvement.
3. Fraction of edges cut.
4. Real system implementation.
5. Page ranking implementation in spark.
6. Page ranking in offline system with small size data.

Conclusion result table analysis graph which heuristics applied better in specific data type,

why different order of streaming for different data set ..... verify original paper our assumption

Future work We would like to find the best performance heuristic for our specific graph type and algorithm, or we could find the difference of each heuristic when applying on various data type and algorithm.