Distributed graph algorithms for husky

Li Shuhe

Introduction

As data collection is more and more easy. Efficient distributed framework is needed. Husky is a new data-parallel computing system which is developed by cuhk team. It has shown high performance for some fundamental algorithms[5]. It can be seen that this framework will be accepted by more and more customers.
As it is a new framework, some specific implementations are in need. The system should be more complete and human friendly. Thus fulling library is a must job. Graph algorithm is part of them. Graph is one of the most common-seen data in real life, whose algorithms are important measurement standard for framework aiming to deal with big data. Of course it can provide lots of information people need. Thus, good implementations of graph for distributed data-computing system are important. My research is based on implementing graph algorithm on husky. In the following section, I will introduce the method of these graph implementations and the performance of them.

Method

The graph is the relationship among vertices. To simplify explanation, we assume every vertex is assigned with unique ID. And all other nodes the vertex points to (we called them successors) have been added into vector nbs.

Connected component

Connected component is one of the most fundamental application for distributed graph computing. It aims to finding the vertices who are connected. One effective algorithm is hash-min[4], which will label all vertices who are connected with same id (to distinguish, we call it compID). The base idea is send the smallest compID to all successors.
At first, the compID of every vertex will be initialized by ID. Then the algorithm is divided into two parts, sending messages and dealing with messages. Sending messages is sending own compID to all successors. Dealing with messages will search the smallest compID among messages and replace own compID with this value if the value is smaller. Recursively executing these two steps, until no compID changed. Finally, the connected component will be labeled. The vertices who are reachable by each other will have same compID, which is the smallest ID of these connected component. The performance will be shown in result section.

Sssp

Single source the shortest path (SSSP) is another fundamental implementation which aims to find shortest path for undirected weighted graph. We use distributed Dijkstra's algorithm, which is very similar to hash-min.
Users should provide source node and destination node. The whole processing is also base on sending messages to successors and dealing with messages. Firstly, we find

the source and label it. Then recursively send weight of edge to successors and label with the smallest costs until there is no new vertex reached. All the vertices which are reachable from source are determined now. We can get the SSSP in reverse way from destination to source node.

Triangle counting

Triangle counting is another implementation for unweighted undirected graph. The number of triangles is a computationally expensive graph statistic which is frequently used in complex network analysis[2]. There exist serval different implementations. The speed and memory consumption will be totally different. The naïve one is just sending all successors to all successors. The vertex that receives the messages will find the intersection of own successors and messages. This implementation will count each triangle 6 times, which is an expensive way.
A simple idea to improve it is to make vertex in order so that we can count each triangle once. For example, when the vertex sends messages to successors, it will only send a vector which includes successors who have higher order to all successors with lower order. Thus, only the lowest order node in one triangle will count.
There still exist another improvement. We can do preprocess by abandoning all successors who have higher order, then just send higher order successors to lower successors. In this way, the whole processing will be a little faster but the idea is same with before.
As it’s a distributed algorithm, to assign same work load to worker is important. Thus we use degree to make order in case of that high degree vertex will waste lots of time when other workers have finished.
As some vertex may have high degree, there may exist situation that the memory consumption is too expensive. To save it, the maximum messages each round should under control. The whole processing will be divided into several rounds.

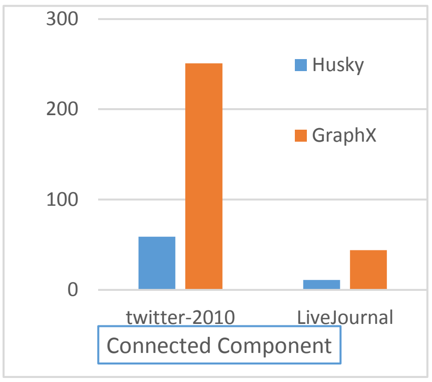
MSF

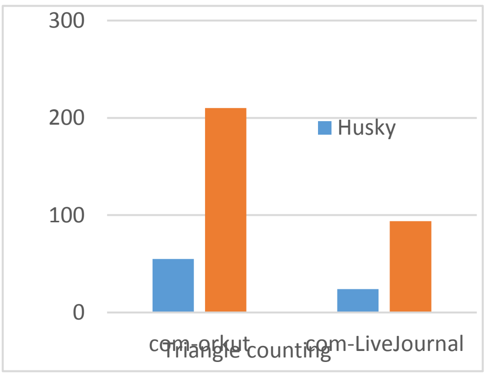
It is simple to find maximum spanning forest (MSF) in single machine. There exist a lot of algorithms for it. But for distributed system, the algorithm is complicated.
The whole processing can be divided into 3 parts. Label vertices, group vertices and delete edges.
Label vertices is to distribute supervertex, subvertex and other types. At first, all vertices are supervertex. Each supervertex will choose an edge of smallest weight among successors. And then send request to the successor this edge point to. Those edges will construct part of MSF. Each part must have one edge who was chosen twice. The vertices this edge connected are new supervertex and subvertex. Other vertices in this subgraph are points.
Group vertices is to mark all vertices who are connected with edges chosen now. We can use connected component algorithm to finish it.
Delete edge is to delete all edges which connect two vertices who have been grouped by before processing. In this way, the next step, supervertexs choose smallest weighted edge will integrate the subgraph. And recursively execute these 3 parts, the

whole MSF will be chosen.

Webgraph
Webgraph is a popular project to compress graph. It can compress the whole graph to less than half size. If we need to use it in distributed system, some algorithms should be changed. As the compression won't store the vertex's id in order to realize high compression ratio. It can't work when we use it in distributed system because the vertex is not in order. I store another extern vector of vertices ids in properties file. The vertices id and the successors will be store in different way.
The base processing is that assign vertices to workers, then each worker will construct subgraph, finally store these subgraphs to HDFS.

Results
I tried to set the same environment when testing those algorithms. All experiments will use one master with 16 workers. The speeds of these graph algorithms are faster compared with other system.
It can be seen that when test connected component algorithm on twitter-2010 and livejournal, the time consumption is much less than GraphX[1]. Triangle counting and other graph algorithm also perform better than Giraph[3].





Data name	Twitter-2010	LiveJournal	Com-orkut	Com-LiveJournal
|V|	41,652,230	4,847,571	3,072,441	3,997,962
|E|	1,468,365,182	68,993,773	117,185,083	34,681,189

Discussion
The distributed graph algorithm is all based on messages transition, which can always be divided into sending messages and dealing with messages. It hard to say that those implementations are best forever, but for now, they perform best. There may exist other improvements, and I am very glad to learn them if they perform better.
Conclusion:
Husky is a great distributed platform for graph. It will cost less and perform better. As for graph, the basic implementation such like connected component, SSSP, MSF, triangle counting are perform good. But the speed is not the only thing. To save memory of disk and make the memory consumption and speed balanced is also

important. As big data is developed day by day, husky will be accepted by more and more persons.

Reference

[1] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica. Graphx: Graph processing in a distributed dataflow framework. pages 599–613, 2014.
[2] M. N. Kolountzakis, G. L. Miller, R. Peng, and C. E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *IM*, 8(1-2):161–185, 2012.
[3] L. Quick, P. Wilkinson, and D. Hardcastle. Using pregel-like large scale graph processing frameworks for social network analysis. In *ASONAM*, pages 457–463, 2012.
[4] D. Yan, J. Cheng, K. Xing, Y. Lu, W. Ng, and Y. Bu. Pregel algorithms for graph connectivity problems with performance guarantees. *PVLDB*, 7(14):1821–1832, 2014.
[5] F. Yang, J. Li, and J. Cheng. Husky: Towards a more efficient and expressive distributed computing framework. *PVLDB*, 9(5):420–431, 2016.