Sheep: A Scalable Distributed Graph Partitioner

Daniel Margo dmargo@eecs.harvard.edu Margo Seltzer margo@eecs.harvard.edu

Problem
Graph partitioning enables efficient computing on very large graphs. But how can we efficiently find partitions of graphs in the absence of an a priori partitioning?

Solution
Sheep is a scalable distributed graph partitioning algorithm with a map-reduce structure. Sheep’s runtime and results are independent of a priori partitions, so the input graph can be arbitrarily distributed among jobs.

Overview
Sheep partitions a graph by:

1. Sorting the vertices,
2. Reducing the graph to an elimination tree,
3. Partitioning that elimination tree, and
4. Translating the result into graph partitions.

Partitioning
Sheep is an edge partitioner that minimizes communication volume. CV measures the number of partitions that each vertex communicates with. Vertices that communicate with more than one partition are called border vertices. When Sheep partitions the tree, it upper bounds the CV by bounding the set of border vertices.

Example: Vertex 7 is the only possible border vertex between the left and right partition sets.

Algorithm
Sheep creates its tree via a union-find algorithm. The tree depends on the graph and the vertex sort order.

Example
Vertex 7 has 5 edges into 3 preceeding union-find sets. It adopts one child for each set. Then, X is below 7 for all edges (X,7).

Pseudocode
Let U = (V,E) be a unifind on a set V that chooses as each subset representative the maximum element in that subset by a total order P = (X,<). Let T = (V,Ti) be an elimination tree on a vertex set V.

Require: G is an undirected graph (V,E)
Require: P is a total order (V,<)
Function: Persistent_union_find(G,P):
U = (V,P)
    T = (V,μu)

For all (x,z) in E, x < z do:
    y = μu(x)
    If y = μu(z) then:
        μu(y,z)
        T1: = T1 u (y,z)

Let T(G,P) be the tree produced by Sheep in order P. Let G1 and G2 be two subgraphs of G such that G1 U G2 = G. Then, we prove that: T(G,P) = T(T1(G1,P) U T(G2,P), P)

Sheep is fast and scalable!
8GB commodity machine with SSD Sheep uses cores efficiently on in-memory graphs. Sheep is up to 6 times faster than METIS on these graphs. Sheep can process out of memory by splitting the graph into working sets.

256GB cluster node with Infiniband 42 million vertices, 1.5 billion edges Scaling is limited by the reduce step. Sheep is more than 8 times faster than Fennel, which takes 22 minutes. METIS cannot process this graph in 256GB.

Sheep finds good partitions!
Sheep’s partitions improve with a better sort order. With a high-quality order, Sheep’s partitions are comparable to METIS. At present these orders are expensive to compute, but this is exciting for future work in e.g. graph database cracking.

References

For more info, please see our paper in VLDB’15! https://github.com/dmargo/sheep