Dealing with Massive Data March 24, 2011

Homework 2: Due April 14, 11:59pm

For each of the problems below you must prove your answer correct. You are encouraged to discuss problems with each other in small groups (2-3 people), as long as you list all discussion partners on your problem set. Discussion of homework problems may include brainstorming and verbally walking through possible solutions, but should not include one person telling the others how to solve the problem. In addition, each person must write up their solutions entirely on their own; you may not look at another student's written solutions. Moreover, all materials you consult must be appropriately acknowledged.

Data

For questions 2 and 3 you will be implementing your algorithm and running it using the AWS framework. Please submit your code together with a readme file describing how we should run it using the courseworks interface. You may hand in the written answers using a hardcopy in class.

The data consists of two real life examples, one being a subset of a webgraph on 700,000 nodes and roughly 7M edges; the other a subset of a social network on 7M nodes and roughly 70M edges. Each graph is present as a file with one edge per line. An edge (u, v) is presented as `u [tab] v` with `[tab]` representing a tab character. The data is located at:

1. `http://coms6998.s3.amazonaws.com/data1.txt`
2. `http://coms6998.s3.amazonaws.com/data2.txt`

You should test your code on smaller examples to ensure correctness before using it on the large datasets.

1. `http://coms6998.s3.amazonaws.com/example1.txt` should return 1
2. `http://coms6998.s3.amazonaws.com/example2.txt` should return 5

Question 1 (MapReduce)

The *prefix-sum* operator takes an array $a_1, a_2, \ldots, a_n$ and returns an array $s_1, s_2, \ldots, s_n$ where $s_n = \sum_{j \leq i} a_j$. For example starting with an array 17 0 5 32 it returns 17 17 22 54. Describe how to implement *prefix-sum* in MapReduce, where the input is stored as $\langle i; a_i \rangle$. That is the key is the position in the array and the value is the value at that position. In your analysis assume that each of your machines has $O(n^c)$ memory for some constant c.

Question 2 (Large Memory CC)

Finding out the number of connected components in a graph is a key subroutine in many graph algorithms. Recall the algorithm presented in class. Given a graph $G = (V, E)$, arbitrarily partition the edges into k non-overlapping groups, $E_1, E_2, \ldots, E_k$ with $E = \cup_i E_i$. In parallel, find the connected components on the graph $G_i = (V, E_i)$ and remove the edges that do not contribute to the connectivity. Finally, combine all of the remaining edges on a single machine and find the connected components.

The pseudocode for the algorithm is as follows. We use $\langle k; v \rangle$ to represent a key, value pair.

Algorithm 1 Connected Components(V,E, k)

Map 1: Input: edge (u, v) // typically represented as $\langle u; v \rangle$.
  Let r be a random integer between 1 and k.
  Output: $\langle r; (u, v) \rangle$.
Reduce 1: Input: set of edges E_i for key i
  Maintain a new graph $F_i = \emptyset$.
  foreach $(u, v) \in E_i$
  if (u, v) are connected in F_i **then**
    continue
  else
    $F_i \leftarrow F_i \cup \{(u, v)\}$
  foreach $(u, v) \in F_i$
    Output: $\langle i; (u, v) \rangle$.
Map 2: Input: edge (u, v)
  Output: $\langle \$; (u, v) \rangle$ //where \$ is a special symbol
Reduce 2: Input: set of edges $\cup F_i$
  Compute $c =$ number of connected components on $H = (V, \cup F_i)$.
  output $\langle \$; c \rangle$.

Implement the algorithm and run it on the two datasets given by the TAs. For each dataset, `data1` and `data2`:

1. (i) Report the total number of connected components
2. (ii) try the algorithm with $k = 5, 10, 20, 50$ partitions. Report the running time for each k.

Question 3 (Small Memory CC)

The algorithm presented in class used $O(|V|)$ memory per reducer. Describe and implement an algorithm that uses $o(|V|)$ memory per reduce instance. That is for *any* input graph, the maximum number of values associated with one key at any time during the execution of the algorithm is less than n. Compare this approach to the algorithm in Question 2. In particular, give the running times for each dataset.