**Parallel Autocomplete**

Eugene Kim
ek3192

**Introduction**

I plan to write a program that parses a corpus of English text for word counts and provides at most K autocomplete suggestions by descending frequency to each input string given by the user.

**Details**

I foresee two main steps to the program:

1. We must obtain word counts from the sample corpus. To find words, all non-alphabetic characters aside from whitespace will be discarded and the remainder will be treated as lowercase. The word counts will be stored in a trie data structure, a 26-ary tree where each node represents a letter and holds a word count (nonzero if it's a complete word, zero if the node is part of a substring).

2. We must provide relevant autocomplete suggestions to user inputs. Each input string given by the user will be traversed through the trie, after which a graph traversal will be performed on all the possible descendants. The possible words/counts will be collected, and the K most frequent words will be outputted to the user in descending frequency.

**Parallelization**

There are opportunities for parallelization in both steps.

1. Instead of gathering word counts from the corpus sequentially, we can parallelize the task by using MapReduce. Multiple mappers will receive distributed fractions of the corpus to create (word, 1) pairs in parallel. After all mappers are done, multiple reducers will take those (word, 1) pairs and obtain (word, frequency) in parallel. These results will be used to populate our trie.

2. The graph traversal to find the word counts of all possible descendants after the input prefix has been traversed can be parallelized as well. The traversal of each descendant at a node can be sparked in parallel. If we proceed in DFS fashion, we can now expect to be bounded in computation by the deepest depth rather than the total number of descendant nodes.

**Evaluation**

We can evaluate the speedup from parallelization in the two steps.

1. There is a valid concern that disk IO will be the primary bottleneck for building the word count trie. I will investigate with ThreadScope in order to ascertain that non-IO computation does indeed get sped up through parallelization through MapReduce.

2. Comparing the speedup in autocomplete queries is straightforward between sequential and parallelized programs. The most meaningful comparisons will be for prefix inputs with a large number of possible autocompletes.

**Data**

This program will have to scale to large inputs, so I am considering using large corpi like the Shakespeare example from wordFreq homework 4 or Wikipedia articles. Also, to simulate search engine autocomplete, I am considering using a corpus containing Google search queries.