Kiara Barias & Sukanya Raj

Project 2 Report

May 17, 2020

Algorithm Design

The following is the algorithm to the first subproblem, which calculates Term Frequency – Inverse Document Frequency. The problem is separated into two phases because the first phase calculates the term frequency, and the second phase calculates term frequency – inverse document frequency. Two different key-value pairs must be used for each phase. The first phase takes in a document and its contents as a key-value pair, and outputs the document id and a term as a key-value pair for each term in the document. The combiner then aggregates all the terms contained in a document and their counts into an associative array. This is implementing the 'stripes' design pattern which makes better use of combiners. Given that the combiner aggregates all the terms and their count into a single array, this takes the job of a reducer and would make the presence of a reducer in this phase redundant and unnecessary. The second phase takes the document and the counts for each term within it, and outputs each term along with its term frequency for the document it appears in. Similarly, to the first phase, the combiner aggregates all the term frequencies for a specific term in a doc into an associative array. Again, this is using the 'stripes' design pattern for a more efficient solution. From this key-value pair, the term frequency- inverse document frequency can be calculated.

Phase 1

Map (using stripes implementation- outputs an associative array)

I: <docid, terms>,...

O: <docid, term>,...

Combiner

I: <docid, term>,...

O: <docid, [term1:count, …]> , ...

Phase2

Map (gets term frequency for term using total terms in that doc and count)

I: <docid, [term1:count, …]> , ...

O: <term, (docid, tf)> , ...

Combiner

I: <term, (docid, tf)>, ...

O: <term, [(docid, tf), …]> , ...

Reduce

I: <term, [(docid, tf), …]>, ...

O: <(term, [docid_tf*log(all_docs/docs_with_term), …]>, ...

Sub problem 2 Algorithm

The following is the algorithm design for sub-problem 2. This problem calculates the cosine similarity between a query term and all terms associated with the TF-IDF matrix. The first phase calculates the similarity score and the second phase sorts those similarity scores in descending order. In the first phase, the input for the map is the precomputed TF-IDF value for each term and it outputs two arrays for all the TF-IDF values for the two terms. The mapper outputs key-value pairs with terms as keys and corresponding associative arrays as values. The reducer takes these values and computes their similarity scores. In the second phase, sorting is implemented in the map. It takes the similarity scores calculated in phase one and outputs those scores in decreasing order. We weren't able to correctly implement the mapper or the similarity scores. Hence, we don't have the top five scores.

Phase 1:

Map (using stripes implementation)

I: <(term, [docid_tf*log(all_docs/docs_with_term), …]>, ...

O: < (q_term, term), (tfidf_value, …) >, …

Reduce

I: < (q_term, term), (tfidf_value, …) >, …

O: < (q_term, term), similarity_score >, …

Phase 2:

Map: (sort similarity scores in descending order)

I: < (q_term, term), similarity_score >, …

O: < similarity_scoreN> , …, <similarity_score1>