COMP9313 - Set Similarity Join Using Spark on AWS

Nihit Vyas (z5140232)

Implementation and Optimization:

The naïve approach tried at first involved using prefix filtering to select candidate pairs for similarity matching. I tried implementing the solution with only prefix filtering but ended up with a high runtime for larger text files.

The final implementation of this project is based on the PPJoin algorithm proposed by Professor Wei Wang, Scientia Professor Xuemin Lin and Associate Professor Chuan Xiao in the paper Efficient Similarity Join for Near Duplicate Detections. The implementation with PPJoin has considerably increased the optimality of the solution, however, since I have not used Spark in the most efficient way, I have not been able to achieve the best running time (20 sec for unsorted.txt but approximately 45 min for flickr_london on 4 nodes cluster). The reason for this is that I have not used broadcast variables and hence the runtime on the clusters has increased and I have observed a significant amount of time is spend on sorting and output file writing.

The runtime on each cluster is given in the Runtime.jpg file. The implementation combines prefix filtering and positional filtering methods, considerable reducing the number of comparisons made to check document similarity.

Following the proposed algorithm, the implementation takes place in three stages with **indexing phase, candidate generation phase** and **verification phase.** In the indexing phase the inverted indices of the tokens that appear in each record are built. The candidate pairs are then generated from the inverted indices. The candidate pairs are those that have a potential to meet the similarity threshold due to the prefix filtering principle. The final Jaccard Similarity is calculated between the candidate pairs in the final verification phase.

The longest possible prefixes of each record are found using the prefix length formula – $\text{prefix length} = |x| - \lceil t \cdot |x| \rceil + 1$. Another filtering step involves positional filtering for which given an ordering O of the tokens is given. As explained by the authors, let token w = x[i], w partitions the record into the left partition xl(w) = x[1 . .(i − 1)] and the right partition xr(w) = x[i . . |x|]. If O(x, y) ≥ α, then for every token w ∈ x ∩ y, O(xl(w), yl(w)) + min(|xr(w)|, |yr(w)|) ≥ α.

The positional filtering is combined with prefix-filtering in PPJoin. The algorithm is described in figure-1. For each record x, ppjoin finds the candidates that intersects with x's prefix and obtains the overlap ubound <- 1 + min(|x|-i, |y|-j) which is an upper bound of the overlap between right partitions of x and y for a current token w. The internal threshold
The generated candidates are further matched using Jaccard Similarity given by: Sim(x,y) = |x ∩ y|/|x∪y|. The authors have also designed an algorithm to optimize the final verification stage by comparing the last token in both prefixes and only the suffix of the record with smaller token needs to be intersected with the entire record. This implementation does not involve the optimised verification algorithm which attributes to a higher runtime.

The final pairs that have a similarity greater than the given threshold are written in the output file.
The runtime for smaller files including unsorted.txt has been found to be within seconds however, flickr_london.txt runs for a longer time which I believe is due to not using broadcast variables.

The program has been successfully run on AWS for the flickr_london.txt and the screenshots are available as attachments. The runtime is lowest on cluster 3 with 4 nodes followed by cluster 2 of 3 nodes followed by cluster 1 of 2 nodes.

Algorithm 1: ppjoin (R, t)

Input : R is a multiset of records sorted by the increasing order of their sizes; each record has been canonicalized by a global ordering $\mathcal{O}$; a Jaccard similarity threshold t
Output : All pairs of records $\langle x, y \rangle$, such that $sim(x, y) \geq t$

```
1  S ← ∅;
2  I_i ← ∅ (1 ≤ i ≤ |U|);
3  for each x ∈ R do
4      A ← empty map from record id to int;
5      p ← |x| − ⌈t · |x|⌉ + 1;
6      for i = 1 to p do
7          w ← x[i];
8          for each (y, j) ∈ I_w such
           that |y| ≥ t · |x| do      /* size filtering on |y| */
9              α ← ⌈(t/(1+t))(|x| + |y|)⌉;
10             ubound ← 1 + min(|x| − i, |y| − j);
11             if A[y] + ubound ≥ α then
12                 A[y] ← A[y] + 1;
13             else
14                 A[y] ← 0;                    /* prune y */;
15         I_w ← I_w ∪ {(x, i)};
           /* index the current prefix */;
16     Verify(x, A, α);
17 return S
```

Figure 1- ppjoin algorithm.