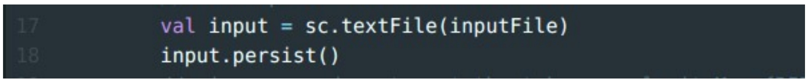
Optimization

Id: z5147182
Name: Xinyuan Qian

Optimization strategies:

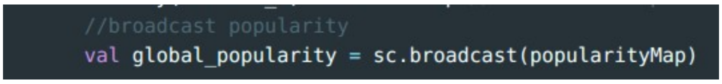
(1) Always operate data on rdd except that I have to collect data in the popularity map.

(2) Persist the input rdd, in which case Spark will keep the elements around on the cluster for much faster access the next time you query it.



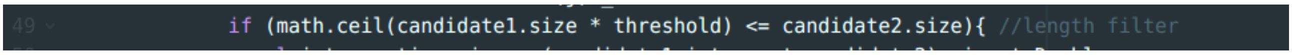
```
        val input = sc.textFile(inputFile)
        input.persist()
```

(3) Broadcast the popularityMap. Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. They can be used, for example, to give every node a copy of a large input dataset in an efficient manner.



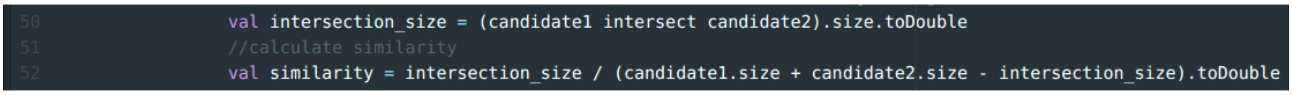
```
    //broadcast popularity
    val global_popularity = sc.broadcast(popularityMap)
```

(4)Apply PPjoin :length filter. For example, for set {A,B,C,D,E} and threshold=0.8, discard all sets with length smaller than 4.



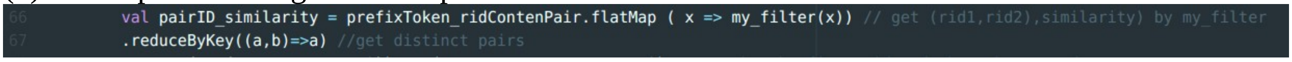
```
                if (math.ceil(candidate1.size * threshold) <= candidate2.size){ //length filter
```

(5) Only calculate intersection instead of both intersection and union when calculating similarity.



```
                val intersection_size = (candidate1 intersect candidate2).size.toDouble
                //calculate similarity
                val similarity = intersection_size / (candidate1.size + candidate2.size - intersection_size).toDouble
```

(6)Use map-reduce to get distinct pairs instead of “.distinct”.



```
        val pairID_similarity = prefixToken_ridContenPair.flatMap ( x => my_filter(x)) // get (rid1,rid2),similarity) by my_filter
        .reduceByKey((a,b)=>a) //get distinct pairs
```

(7) I tried Positional Filter but the result is slower than before. It is probably because the data in filckr_london.txt are not suitble for it. Extra calculating time is more than the time saved.