Big Data Analysis
Group 6

Cedric Mingneau (1232611)
William Thenaers (1746077)
2019-10-22

Similar Items: Plagiarism Detection

Initial analysis

1. *How can we establish plagiarism in python source files? Keep in mind that students often tend to change solely the names of variables and functions within the code.*

   - In de eerste plaats *tokenizen* we de source files. Met behulp van een Python language parser identificeren we woorden uit een source file die de naam van een variabele moeten voorstellen. Op die manier kunnen alle variabelen, klassen- en functienamen vervangen worden door "ID"-tokens. We zouden deze tokens dan als één karakter kunnen voorstellen (bijv. "$"). Zo voorkomen we dat de gelijkenis van de documenten negatief beïnvloed wordt door variabelen anders te noemen.

   - Eens de broncode uniforme ID's bevat, kunnen we deze opsplitsen in woorden van een bepaalde lengte die we vervolgens verder in k-shingles samenvoegen.

     Vb "`my_list = { x * x for x in range(10) }`" => "`$={$*$for$inrange(10)}`"
     => 5-shingles: {`$={$*`, `={$*$`, `{$*$f`, `$*$fo`, `*$for`, …, `(10)}`}

2. *Explain how Minhash signatures can be computed in a MapReduce environment and argue about the correctness of your approach. Describe in detail how hashes are computed.*

   - Indien alle bestanden gelijkmatig als chunks worden verdeeld over enkele compute nodes, kan het parallel berekenen van MinHashes voor problemen zorgen. De nodes kunnen vlot de tokenizing- en shingling-stappen uitvoeren. Bij de chunk-aanpak verliezen we echter de bundeling van een document. Je kan de MinHashing-stap enkel uitvoeren op bit-vectoren die op het volledige document zijn gebaseerd. Om de shingles naar behoren te hashen, zouden we daarom eerst alle chunks moeten defragmenteren in documenten. Deze stap lijkt ons omslachtig.

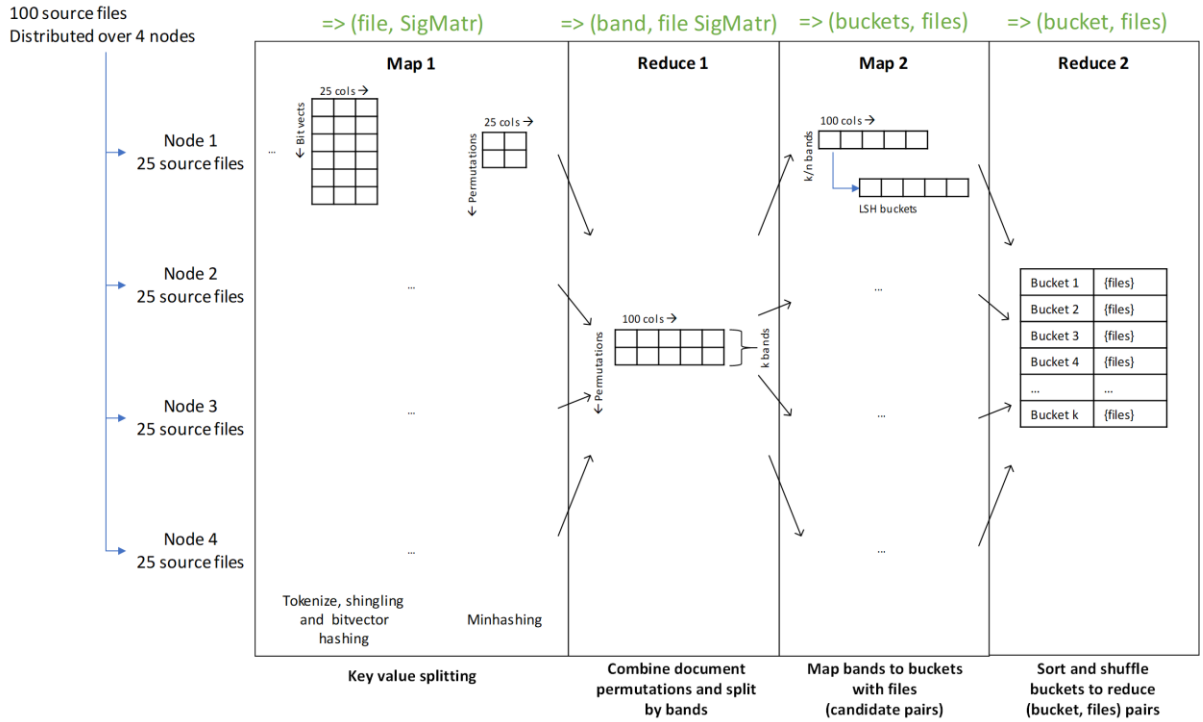
   - Vermits het ontdekken van plagiaat wordt gebruikt binnen de context van schooltaken, lijkt het ons veroorloofd om te veronderstellen dat de documentgroottes niet erg variabel zijn. Dit betekent dat we documenten over nodes kunnen verdelen zonder een echt ongelijkmatige werkverdeling te veroorzaken. Alle voorgenoemde stappen kunnen dan op de nodes gebeuren zonder de omslachtige defragmentatiestap. Omdat alle documenten 'heel' blijven zullen de MinHash signaturen correct worden berekend.

   - Om een goede Signature Matrix te bekomen, moeten we een aantal pseudo-random permutaties uitvoeren op de rij-(/bit-)indices van de gehashte bitvectoren van shingles. Deze permutaties worden gedaan via een aantal hashfuncties H die de rijindices afbeelden op andere indices. Zulke hashfuncties kunnen bijvoorbeeld de vorm hebben van $(\mathrm{ax} + \mathrm{b}) \,\%\, \mathrm{length}$, met a en b distinct verschillende getallen voor elke functie en length het aantal shingle-bits in de document input matrix.

Cedric Mingneau (1232611)
William Thenaers (1746077)

2019-10-22

3. *Present the end-to-end architecture of the pipeline, preferably by means of a visualization. Draw specific attention to the input and output of each of the MapReduce stages involved. Clearly indicate the key used during reduce phases and describe the logic of non-trivial map operations.*



1. De bronbestanden worden gelijkmatig verdeeld over de beschikbare compute nodes.
2. Elk bestand wordt gestreamd door een tokenizer die identifiers vervangt door bvb. `$` en whitespace negeert. N-shingles worden vervolgens opgesteld en gehasht naar een bitvector waarin elke bit aanduidt of de shingle uit de shingle-set voorkomt in het document.
3. Minhashing wordt toegepast en de Signature Matrix wordt opgesteld afhankelijk van een aantal pseudo-random gekozen hashfunctie, en dit voor elk document op de node.
4. In de volgende stap worden de Signature Matrices “samengevoegd” om voor elk bestand enkele banden uit de bitvector te kiezen voor verdere behandeling op elke node.
5. In deze banden worden mogelijke kandidaatparen beschouwd en de gelijkenis berekend. Paren waarvoor de kans op gelijk zijn boven een bepaalde grens ligt, worden naar eenzelfde bucket gehasht (Locality Sensitive Hashing).
6. In de voorlaatste stap worden de buckets van elke node opnieuw samengevoegd om een unieke set te creëren van elke file in elke bucket.
7. Ten slotte kan men de lijsten uit elke bucket nu samenvoegen om tot het finaal resultaat te komen.

In stap 1-3 worden *(file, Signature Matrix)*-paren aangemaakt in de mapping. Stap 4 maakt hiervan nieuwe *(band, file Signature Matrix)*-paren, waarbij een aantal banden per node verdeeld worden, samen met elke kolom (elke file). Stap 5 zorgt vervolgens dat deze files naar *(bucket, files)*-paren worden omgezet, die in stap 6 over elke node gelijkmatig verzameld worden, waarbij node 1 bijvoorbeeld $buckets/4$ aantal buckets krijgt die hij moet samenvoegen. In de laatste stap produceren we dan opnieuw *(bucket, files)*-paren, maar deze keer werden de lijst met bestanden reeds samengevoegd in de vorige stap. Elke bucket bevat een indicatie van de waarschijnlijkheid dat de bestanden in die bucket gelijkwaardig waren. Deze files kunnen vervolgens in meer detail vergeleken worden om na te gaan wat er precies in voorkomt dat gelijkaardig was aan de andere bestanden in diens bucket.