Практические задания 3 и 4

Курносов Михаил Георгиевич

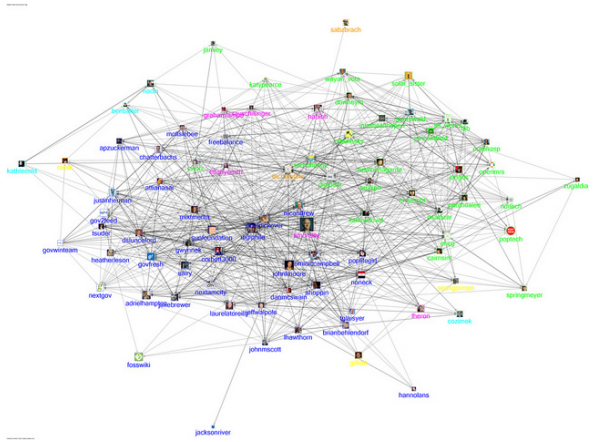
E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Курс «Распределенная обработка информации»
Сибирский государственный университет телекоммуникаций и информатики
Весенний семестр, 2023

Анализ графа пользователей Twitter

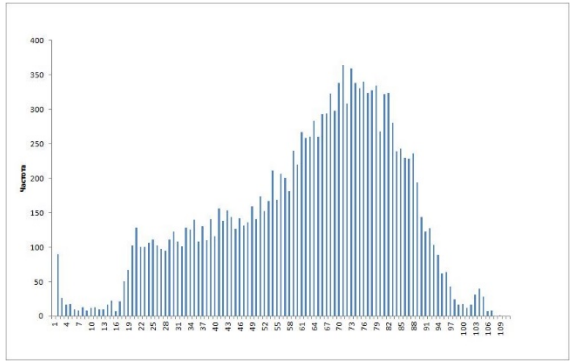
- Задан граф пользователей Twitter с информацией о их подписчиках
- Граф хранится в текстовом файле, каждая строка которого имеет следующий формат:

  <user_id> <follower_id>



- Требуется вычислить распределение количества подписчиков (статистическое распределение выборки)
- Определить Top50 пользователей по числу подписчиков
- Вычислить среднее количество подписчиков
- `hdfs dfs -cat /pub/followers.db`

```
1804299383 174900922
1804299383 1180726932
1804299383 1527279874
```



Анализ графа пользователей Twitter

- **Вычисление распределения количества подписчиков** (статистическое распределение выборки)

$nfollowers_1$	$nfollowers_2$	...	$nfollowers_k$
$freq_1$	$freq_2$	...	$freq_k$

- Вычисляем число подписчиков каждого пользователя
  - **map**(user_id, follower_id) -> (user_id, follower_count) // In-mapper combiner
  - **reduce**(user_id, [follower_count]) -> (user_id, follower_count)

- Распределение числа подписчиков (*подготовить данные с предыдущего шага*)
  - **map**(user_id, follower_count) -> (follower_count, user_count)
  - **reduce**(follower_count, [user_count]) -> **(follower_count, user_count)**

Анализ графа пользователей Twitter

- **Вычисление Top50 пользователей**

- Вычисляем число подписчиков каждого пользователя
  - **map**(user_id, follower_id) -> (user_id, follower_count) // In-mapper combiner
  - **reduce**(user_id, [follower_count]) -> (user_id, follower_count)

- **Top50**
  - **map**(user_id, follower_count) -> top50[(user_id, follower_count)]
  - **reduce**([(user_id, follower_count)]) -> top50[(user_id, follower_count)]

Анализ графа пользователей Twitter

- **Вычисление среднего числа подписчиков**

- Вычисляем число подписчиков каждого пользователя
  - **map**(user_id, follower_id) -> (user_id, follower_count) // In-mapper combiner
  - **reduce**(user_id, [follower_count]) -> (user_id, follower_count)

  - **map**(user_id, follower_count) -> (user_count, follower_count)
  - **reduce**([(user_count, follower_count)]) -> (follower_count_avg, null)

Оптимизации

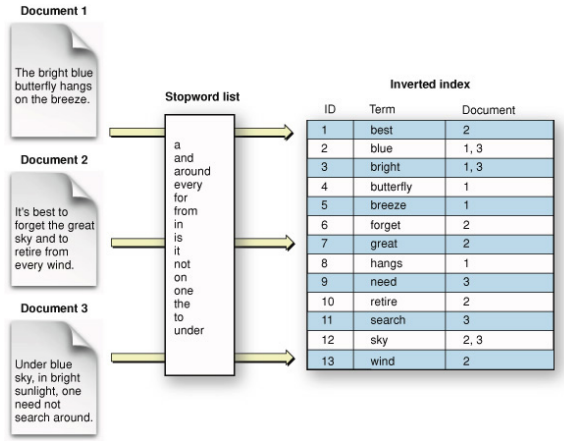
- Combine
- Использование счетчиков

Инвертированный индекс Wikipedia

- Имеются дампы Википедии в формате XML (http://dumps.wikimedia.org)

- Требуется построить инвертированный индекс (Invertex index) для русской Википедии (и английской — по желанию)
- Результирующий инвертированный индекс должен иметь следующую структуру: (word, [<docid1, TF-IDF1>, <docid2, TF-IDF2>, …])

- Статьи должны быть отсортированы в порядке убывания TF-IDF
- Для каждого слова ограничить список статей *N* наиболее релевантными
- Определить и исключить из индекса Top20 высокочастотных слов



Инвертированный индекс Wikipedia

- Результирующий инвертированный индекс должен иметь следующую структуру: (word, [<docid1, TF-IDF1>, <docid2, TF-IDF2>, ...])
- Статьи должны быть отсортированы в порядке убывания TF-IDF
- *TF*(*t*, *d*) — это число вхождений слова *t* в документ *d* (Wiki-статью)
- *IDF*(*t*, *D*) — обратная частота, с которой слово *t* встречается во множестве документов *D*

$$\mathrm{IDF}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

Инвертированный индекс Wikipedia

- **Входные пары: (docid, content)** –
  XML: номер (<id>key</id>) и текст Wiki-статьи (<text>value</text>)
- **Определение Top20 высокочастотных слов**
  - **Вариант 1:** 2 задания (WordCount + Top)
  - **Вариант 2:** 1 задание + bash (sort –nr –k KEYDEF | head –n 20)
- **Подсчет количества документов (|D|)**
  - **Вариант 1:** отдельное задание:
    map(docid, content) -> (docid, 1) -> reduce(docid, [1, …]) -> (ndocs, null)
  - **Вариант 2:** статистика (Map input records) или счетчик при определении Top20 слов
  - **Вариант 3:** одновременно с построением индекса

Инвертированный индекс Wikipedia

- **Построение инвертированного индекса**
- **map**(docid, content) -> (word, <docid, tf>)
  - ❑ Фильтруем слова из списка Top20
  - ❑ В ассоциативном массиве накапливаем статистику по всем словам статьи и подсчитываем TF (число вхождений слова *t* в документ)
- **reduce**(word, [<docid, tf>]) -> (word, top*N*[<docid, tf-idf>])
  - ❑ Подсчитываем все статьи с данным словом, запоминая первых *N* статей по TF (PriorityQueue, TreeSet)
  - ❑ Вычисляем число D’ (D’ > 0) вхождения слова *t* во все документы
  - ❑ IDF(t) = log(D/D’)
  - ❑ TF-IDF = TF * IDF
  - ❑ Выдаем результирующую строку индекса: (word, [<docid1, TF-IDF1>, <docid2, TF-IDF2>, …])

$$\mathrm{IDF}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

Инвертированный индекс Wikipedia

- Альтернатива
  - ❑ map(docid, content) -> (<word, tf>, docid)
  - ❑ Partitioner: распределение по word
  - ❑ Comparator: сортировка по word + tf в убывающем порядке
  - ❑ Reduce: запомнить первые 20 статей

$$\mathrm{IDF}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$