Отчет по домашнему заданию 3

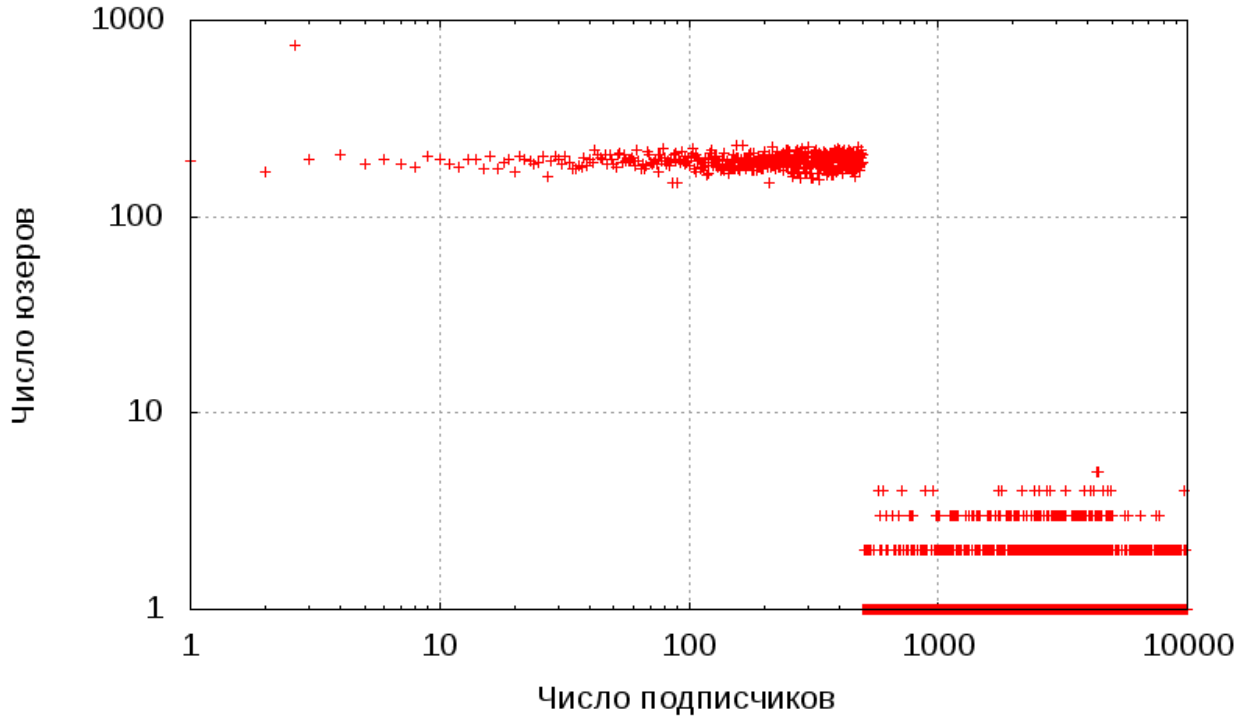
Капралов Степан Алексеевич

Задача 1. Анализ графа пользователей Twitter

Расчеты всех пунктов задания можно легко выполнить имея входной файл типа:
<user_id> <follower_count>
Приведем исходный файл followers.db к данному формату и будем считать это нулевым шагом задания. Для выполнения данного шага возьмем пример WordCount из лекции, и считать будем не все слова, а только поле <user_id>. Выходной файл разместим в hdfs под именем twitter_in_stat.

Распределение количества подписчиков

Статистическое распределение числа подписчиков, посчитаем при помощи того же WordCount, только считать будем уже столбец количества подписчиков в twitter_in_stat. Используя получившийся файл построим график:



По графику видно, что подписчики распределяются не равномерно и от 500 до 10000 подписчиков имеют единицы юзеров, когда как число от 1 до 500 подписчиков распределяется в интервале 100 -- 1000 юзеров.

Определить Top50 пользователей по числу подписчиков

Идея решения заключается в том, чтобы maper вернул пару <follower_count><user_id>, тогда на вход reducer придет отсортированная последовательность по follower_count и ему достаточно будет вывести 50 первых значений. Однако стандартный порядок сортировки ключей между map и reduce определен - от меньшего к большему. Переопределим SortComparatorClass на собственный ReverseComparator класс, в котором обратим результаты сравнивания входных значений. После выполнения программы получим топ 50 пользователей:

N	followers	ID
1	9998	761176847
2	9995	1514756796
3	9987	1267784969
4	9985	1741431984
5	9982	1403811530
6	9970	1154494545
7	9968	663627571
8	9961	114479825
9	9959	1773933401
10	9957	1026345530
11	9950	1598870385
12	9943	1078023685
13	9935	1602654567
14	9932	129001610
15	9922	41521470
16	9922	787406737
17	9915	446095342
18	9914	692788116
19	9893	808940430
20	9889	721227519
21	9881	1548167412
22	9875	1443059210
23	9867	1862171669
24	9851	950130185
25	9842	105483455
26	9838	1919052445
27	9822	2028971856
28	9821	1477149620
29	9811	467588311
30	9803	746100793
31	9801	1605921382
32	9799	951037820
33	9785	848121481
34	9776	1074739904
35	9776	1540426485
36	9774	594521840
37	9769	1926415457
38	9764	1740913352
39	9763	989240775
40	9744	1861549356
41	9739	1241895627
42	9738	389252577
43	9731	839170186
44	9726	123925623
45	9722	365997523
46	9719	1644446862
47	9710	1361386226
48	9708	1221015857
49	9707	10614139
50	9701	179391234

Вычисление среднего количества подписчиков

Подсчитаем сумму числа подписчиков (sum) и количество суммирований (count), в каждом maper. Воспользуемся методом cleanup, что бы передать в фазу reduce единственную пару ключ-значение <sum><count>. В reducer суммируем пришедшие

значения сумм и счетчиков из каждого maper. И в методе cleanup находим среднее значение (sum / count). Получим среднее число подписчиков равное - 400,14.

Задача 2. Инвертированный индекс Wikipedia

Решение выполнено за две map-reduce задачи.

Первая задача:
Стадия map. XmlInputFormat возвращает id статьи и ее содержимое, значит статья целиком находится в памяти. Используя это можем подсчитать индекс TF, так как он локален для статьи. TF = t / d , где t - число вхождений слова в статью, d - число слов в статье. Одновременно с индексом выполним подсчет статей в документе (P). Стадия map вернет следующую пару <word><id@TF>

Стадия reduce. Получим значение P из стадии map путем переопределения метода setup. К value добавим значение P, что бы передать его в следующую задачу. Стадия вернет пару <word><id@TF@P>

Вторая задача:
Стадия map. Просто передает в reduce пару <word><id@TF@P>.

Стадия reduce. По заданию для каждого слова нужно вывести N (в моей программе N = 10) наиболее релевантных статей и отсортировать их по TF-IDF. Индекс IDF = *log*(P / wp), где P - число статей во всем документе, wp - число статей где употребляется искомое слово. TF-IDF = TF * IDF. Индекс IDF в рамках одного слова будет равным для всех документов. Значит для подсчета топа релевантных документов нам достаточно будет знать только индекс TF. Подсчитаем топ одновременно с подсчетом wp используя TreeMap. В результате после прохода списка значений в reduce, мы получим отсортированный топ N документов в TreeMap. Считаем IDF. Выводим топ одновременно вычисляя TF-IDF для каждой статьи. Такой способ позволяет избежать проблемы двойного обхода значений в reduce, с которой можно столкнуться выполняя данное задание. Стадия вернет <word><id@TF-IDF … id@TF-IDF>, value отсортировано по убыванию TF-IDF. Можно задавать несколько стадий reduce, так как при подсчете стадии все вычисления выполняются опираясь только на пришедшие ключ и значения.

Эти две задачи можно объединить в одну, так как в первой бездействует стадия reduce, а во второй стадия map. Но для удобства отладки и тестирования было решено оставить две задачи. Так же можно выполнить их сцепление, что бы сразу подать выходные данный первой задачи на вторую.

Отладочное тестирование проводилось на XML-дампе русской Википедии за один месяц (~ 126 МБ). Путь к файлу на кластере /home/students/pdc2015/pdcuser48/MR/wiki_test, в hdfs в домашнем каталоге pdcuser48. Ссылка http://dumps.wikimedia.org/ruwikinews/20150424/ruwikinews-20150424-pages-articles.xml.bz2

При тестировании на /pub/ruwiki.xml в первой задаче использовался один reduce, во второй 4.