MapReduce 100500.0. WebRobot

Vanya dasfex Khodor

14 мая 2020 г.

Основная логика запуска содержится в mapreduce.sh, в который требуется передать несколько параметров:

1. Входной файл со списком сайтов.
2. Выходной файл, который создастся/перезапишется.
3. Имя для файла, который будет содержать лог ошибок(создастся/перезапишется).
4. Имя для папки с временными файлами, чтобы избежать пересечения имён с имеющимися файлами. После работы скрипта папка будет удалена(да, я решил скинуть решение этой проблемы на пользователя).
5. Глубина, на которую разрешается заход.
6. Кол-во мапов, на которые вы хотите делить ваш вход.

Т.е. запуск может выглядеть так:

```
./mapreduce.sh i1.txt o1.txt log tmp 2 10
```

Логика работы:

1. Изначально копируем содержимое файла во временный файл, который содержится в папке из \$4.
2. Далее в цикле по $i = \overline{1, \$5}$ делаем операции map и reduce.
3. После цикла делаем mapNormalize.
4. Удаляем временные файлы.

map обходит последовательность ключей, выписывает в выходной файл сайт, который мы обошли только что, при этом помечая, что он обойдён, а также все новые сайты, не помечая их, как обойдённые. Тут как параметры в фреймворк mapreduce нужно передать map(то, что запускаем), путь к скрипту, файл с входными данными, файл для выхода, кол-во мапперов, на которые поделить все данные.

Перед reduce происходит сортировка. Позже все ключи делятся на файлы(в одном файле равные ключи, причём равны ли они, определяется с помощью user-defined comparator), и на каждом файле запускается свой reduce. Если хотя бы один ключ в файле помечен, как тот,

который обошли, то на выходе сайт будет помечен обойдённым. Тут в фреймворк передаём reduce, путь к скрипту, вход, выход, путь к компаратору(который возвращает 0, если ключи не равны, и любое другое значение иначе).

mapNormalize запускается один раз в самом конце, чтобы убрать метки у сайтов, которые посещены(и никто не понял, что да как мы вообще делали).

Рассмотрим на примере.

Пусть у нас имеется такой словарь (сайт: сайты, которые мы можем встретить на нём): (A: B, C, D; E: A, B, G).

Входной файл:

```
A \t
E \t
```

После map мы *можем* получить:

```
A \t +
B \t
C \t
D \t
E \t +
A/ \t
B \t
G \t
```

После будет произведена лекикографическая сортировка:

```
A \t +
A/ \t
B \t
B \t
C \t
D \t
E \t +
G \t
```

После разбиваем по ключам:

```
A \t +
A/ \t
---------
B \t
```

```
B \t
---------
C \t
---------
D \t
---------
E \t +
---------
G \t
```

После reduce и мерджа получим:

```
A \t +
B \t
C \t
D \t
E \t +
G \t
```

Была такая проблема, что были адреса вида:

```
A
A/
```

Это плохо как минимум потому, что придётся обходить один и тот же сайт дважды(что долго). Потому я сделал comparator.py для сравнения ключей, который считает ключи равными в случае, если они отличаются не более чем одним слешем в самом конце адреса.