易观大数据算法

yiguan

易观大数据算法

思想

主要技术点采用了 HDFS作为存储， Spark作为数据预处理和核心过滤算法实现。

整个解决方案分为两大部分：

- 数据的预处理
- 查询算法实现

1.数据的预处理

1. 在数据预处理阶段，我们按用户进行了聚合，去除了对于我们无用的字段，将每行作为一个用户的操作集合，按时间戳排序。
2. 时间上统一减去了起始时间（20170501的UTC 时间），并除以10，将long型变为了Int型存储。
3. 操作id 只保留了后两位，这已经可以作为唯一标识。

数据预处理的核心代码

```
var data =
      sc.textFile(args(0))
        .map(_.split("\t+"))
        .filter(_.length > 5)
        .map(items => {
          val userId = items(0)
          val timestamp = ((items(1).toLong - 1493568000000L)/10).toInt
          val opId = items(2).substring(2).toInt
          val params = items(4)
          (userId, timestamp, opId, params)
        })

    //    预处理数据
    data
      .groupBy(_._1)
      .map((x) => {
        var rs = ""
        x._2.toArray.sortBy(_._2)
          .foreach((item) => {
            rs += item._3 + "|" + item._2 + "|" + item._4 + "|"
          })
        rs
```

```
24        })
25        .saveAsTextFile(args(1))
```

经过数据预处理，原始的数据文件，变成了以每行为一个用户的操作集合，并按照时间进行了升序排序。

最终一个原始数据如下的文件

userId	timestamp	opId	detail	date
id1000001	1493584373226	10006	收藏商品 {}	20170501
id1000001	1493650478627	10010	订单付款 {}	20170501
id1000001	1493649968308	10004	浏览商品 {"brand": "Hair", "price": 8200}	20170501
id1000001	1493652617901	10009	生成订单 {"order": 14500}	20170501

经过数据预处理之后，变为了

`opId(后两位)|预处理时间|json串` （循环）

```
1
2 6|1637322|{}|4|8196830|{"brand": "Hair", "price": 8200}|10|8247862|{}|9|8461790|{"order": 14500}|
3
```

至此，数据预处理阶段完成。

2. 查找算法

- 算法思想：

```
1
2
3 1. 算法思想：借鉴最长递增子序列的存储和更新思想
4
5  存储：只存储对应长度的最新时间戳
6
7  更新：先判断该元素能完成的最大长度，在判断对应位置是否需要更新。
8
9 2. 存储结构：
10
11 一个长度为step的数组step（例如ABCDE的step为5），这里为了使step[1]可以和长度为1的结果对应，所以step长度初始为6。
12
13 step[1]存储的为能构成A序列的最新A的时间戳，step[2]存储的为能构成AB序列的最新A的时间戳，依此类推，ABC,ABCD,ABCDE，step[]末尾一旦填值，则直接退出此user。
14
15 3. 具体示例：
16
17 以寻找ABCD步骤为例，初始化数组step[]长度为5。
18
19 (1)   起始分为两种情况，此时step为空：
20
21 找到A，时间戳为tA1：序列A的最新A的时间戳为tA1,step[1]=tA1
22
23 找到BCD：由于step[1]处没有值，无法构成相应序列AB,ABC等，跳过。
```

```
24
25 (2)   接下来，分为三种情况，此时step[1]=tA1：
26
27 找到A，时间戳为tA2：此时此序列A的最新时间戳为tA2，更新step[1]=tA2
28
29 找到B：判断B的时间戳是否和step[1]处的时间戳tA1满足时间窗口，满足，就令step[2]=tA1,表明构成AB序列的最新A的时间戳为tA1,不满足跳过
30
31 找到CD:此时step[2]处没有值，无法构成ABC,ABCD，跳过。
32
33 (3)   接下来分为四种情况，此时step[1]=tA1,step[2]=tA1:
34
35 找到A（tA3）：此时构成序列A的最新时间戳为tA3，更新step[1]=tA3
36
37 找到B：判断与step[1]处时间戳是否满足窗口，满足的话以step[1]处的时间戳更新step[2]处的时间戳。（若在之前没找到新的A，则对这个B和预先存储的tA1判断，结果为构成AB的最新A的时间戳为tA1，更新step[2]=tA1，相当于没更新。若在之前找到的新的A（总体步骤是ABAB），那么step[1]和step[2]中存的都是第一个A的时间戳，此时第三个A（tA3）到来，更新step[1]=tA3，此时step[1]=tA3,step[2]= TA1,代表A序列最新A时间戳为tA3，AB序列最新A时间戳为tA1。此时第四个B到来，与step[1]中时间戳tA3判断时间窗口，如果满足，则对于这个B，构成AB序列的最新A时间戳为tA3,比step[2]的tA1靠后所以更新step[2]=tA3）。
38
39 找到C：与step[2]处的时间戳判断能否满足时间窗口，如果满足更新step[3]=step[2],不满足跳过。
40
41 找到D：没有step[3]无法构造成ABCD,跳过。
42
43 以此类推。
44
45 （4）. 算法结束条件有两种： 1. 所有数据遍历完成。 2.step[length-1]处填值，完成了最终转化。
46
```

- 采用spark来实现算法思想
    - 通过累加器来统计每一个opId的人数。
    - 对每一行，也是每一个用户，采用查找算法求得其完成转化的深度。
    - 输出最终结果。
- 针对本次比赛的spark调优
    - 在正式采用算法处理数据时，先进行了count操作，这是spark的一个action算子，会提前将数据加载到内存中。 这个时间并未纳入最后的时间统计中。这点特此说明（赛前测试，跑测试用例时，提前将数据加载至内存中平均会加快1.5秒--2.5秒）
    - 针对 Spark的 `num-executors`, `executor-memory`, `executor-cores`等参数进行测试与调试。