技能赛命题 1

基于 Hadoop Map-Reduce 的日志统计分析

任务 1

1.1 算法概要

该任务是要统计状态码的总数和各个时间窗口的状态码总数，所以我们在 Map 端处理文本行，从文本行中提取“时间”和“状态码”两个信息，再在 Reduce 端将所有的 Map 输出结果进行汇总和累加，从而得到总的访问频次。为了减少网络 I/O 传输，我们使用 Combiner 执行“本地 reduce”操作，极大地提高了任务执行效率。

1.2 Map-Reduce 模型

1.2.1 分区、排序

- 我们使用默认的 HashPartitioner 分区算法，根据接口字符串的哈希码对 reduce 的个数取模进行分区。但由于本程序使用的是复合键，所以对 hashCode 方法进行了重写。
- 按照时间窗排序

1.2.2 Map 端

- 从文本中的每一行中提取“访问时间”和“状态码”， 并将访问时间作为输出键，将访问时间，状态码，访问次数封装成一个 StatusInterface 类，作为输出值。由于每一行文本只访问了一次，所以这里的总次数都为 1。
- 对于提取过程中“在处理不合理的数据时所抛出的异常”进行捕获并直接 return，这样可以过滤掉不合理的数据，使程序继续执行。

1.2.3 Combiner

- 执行一次本地 reduce 操作，输入键为上面提到的访问时间，输入值为 StatusInterface 类，输出和输入一致。将访问时间和状态码相同的所有键值对的“值中的总频次”累加并输出和。

1.3.4 Reduce 端

- 对于所有 Map 输出结果进行累加求和。因为每个 Map 只是统计了“部分和”，所以在 Reduce 端需要将这些“部分和”进行累加。
- 由于是按照接口分组，所以在一个 reduce 方法中存在不同时间窗的数据，所以我们在 for 循环遍历过程中，利用一个数组存取不同的时间窗，从而可以统计每个时间窗的总频次。

任务 2

2.1 算法概要

该任务是要统计每个 IP 的访问总频次和各个时间窗口的访问频次，所以我们在 Map 端处理文本行，从文本行中提取“IP”和“时间窗”两个信息，再在 Reduce 端将所有 Map 输出结果进行汇总和累加，从而得到总的访问频次。为了减少网络 I/O 传输，我们使用 Combiner 执行“本地 reduce”操作，极大地提高了任务执行效率。

2.2 Map-Reduce 模型

2.2.1 分区、排序、分组

- 我们使用默认的 HashPartitioner 分区算法，根据接口字符串的哈希码对 reduce 的个数取模进行分区。但由于本程序使用的是复合键，所以对 hashCode 方法进行了重写。
- 我们使用二次排序，先按照 IP 排序，再按照时间窗排序
- 我们按接口进行分组。因为考虑到不仅要统计各个时间窗每个 IP 的访问频次，还有统计每个 IP 访问的总频次，所以需要将一个接口的所有键值对作为一组进行处理。

2.2.2 Map 端

- 从文本中的每一行中提取“IP”和“访问时间”，并将 IP 和访问时间封装成一个 IPTime 类，作为输出键，将访问时间和总次数封装成 IpInterface 类作为输出值。由于每一行文本只访问了一次，所以这里的总次数都为 1.
- 对于提取过程中“在处理不合理的数据时所抛出的异常”进行捕获并直接 return，这样可以过滤掉不合理的数据，使程序继续执行。

2.2.3 Combiner

- 执行一次本地 reduce 操作，输入键为上面提到的 IpTime 类，输入值为 IpInterface 类，输出和输入一致。将 IP 和访问时间相同的所有键值对的“值中的总频次”累加并输出和。

2.2.4 Reduce 端

- 对于所有 Map 输出结果进行累加求和。因为每个 Map 只是统计了“部分和”，所以在 Reduce 端需要将这些“部分和”进行累加。
- 由于是按照接口分组，所以在一个 reduce 方法中存在不同时间窗的数据，所以我们在 for 循环遍历过程中，利用一个数组存取不同的时间窗，从而可以统计每个时间窗的总频次。

任务 3

3.1 算法概要

该任务是要统计每个接口的访问总频次和各个时间窗口的访问频次，所以我们在 Map 端处理文本行，从文本行中提取“接口”和“时间窗”两个信息，再在 Reduce 端将所有 Map 输出结果进行汇总和累加，从而得到总的访问频次。为了减少网络 I/O 传输，我们使用 Combiner 执行“本地 reduce”操作，极大地提高了任务执行效率。

3.2 Map-Reduce 模型

3.2.1 分区、排序、分组

- 我们使用默认的 HashPartitioner 分区算法，根据接口字符串的哈希码对 reduce 的个数取模进行分区。但由于本程序使用的是复合键，所以对 hashCode 方法进行了重写。
- 我们使用二次排序，先按照接口排序，再按照时间窗排序
- 我们按接口进行分组。因为考虑到不仅要统计各个时间窗每个接口的访问频次，还有统计每个接口访问的总频次，所以需要将一个接口的所有键值对作为一组进行处理。

3.2.2 Map 端

- 从文本中的每一行中提取“接口”和“访问时间”，并将接口和访问时间封装成一个 URLInterface 类，作为输出键，将访问时间和总次数封装成 TimeCount 类作为输出值。由于每一行文本只访问了一次，所以这里的总次数都为 1.
- 对于提取过程中“在处理不合理的数据时所抛出的异常”进行捕获并直接 return，这样可以过滤掉不合理的数据，使程序继续执行。

3.2.3 Combiner

- 执行一次本地 reduce 操作，输入键为上面提到的 URLInterface 类，输入值为 TimeCount 类，输出和输入一致。将接口和访问时间相同的所有键值对的“值中的总频次”累加并输出和。

3.2.4 Reduce 端

- 对于所有 Map 输出结果进行累加求和。因为每个 Map 只是统计了“部分和”，所以在 Reduce 端需要将这些“部分和”进行累加。
- 由于是按照接口分组，所以在一个 reduce 方法中存在不同时间窗的数据，所以我

们在 for 循环遍历过程中，加入 if 判断以截取不同的时间窗，从而可以统计每个时间窗的总频次。

3.4 优化

考虑到适当增加 Reduce 的个数可以提高程序的执行效率，而在少量数据集上测试时发现接口的种类数为 18 类，而我们又是按照接口进行分区的，所以我们将 Reduce 的个数设置为 18，这样充分提高了并行执行效率。

任务 4

4.1 算法概要

该任务的算法与任务 3 类似，故此处不再赘述。唯一不同的是，该任务需要统计平均响应时间，所以我们在任务 3 的基础之上增加“响应时间”的累加操作，即不仅是访问频次的统计，又多了响应时间的统计，最后用总的响应时间除以总的访问频次即可。

4.2 Map-Reduce 模型及其优化

该任务只是在 Reduce 端相对于任务 3 增加了“除法操作”，其它的以及优化操作基本相同，故此处不再赘述。

任务 5

5.1 算法概要

该任务是在任务 3 的基础上，根据统计得来的 15 天的数据预测第 16 天的数据。只不过将秒时间窗改为了小时时间窗，分组按照小时分组，在同一分组中有不同的接口和不同的天数，因为数据是经过排序的，所以我们很容易利用“for 循环+if 判断”在 Reduce 中统计得到某一接口在该小时时间窗中每天的访问频次，从而对于一种“接口+小时”组合得到一组 15 个数组成的序列，然后根据这 15 个数预测第 16 个数。统计算法同任务 3，此处不再赘述。而对于预测算法，我们使用“灰色系统理论”中的“灰色预测模型”算法来进行预测。

5.2 灰色预测算法

考虑到实际数据只有 15 个，数据量较少，所以采用灰色预测算法，利用生成数来进行预测。

5.2.1 原始数据序列

$$x^{(0)}=\{x^{(0)}(1),x^{(0)}(2),...,x^{(0)}(15)\}$$

5.2.2 生成数序列

$$x^{(1)}=\{x^{(1)}(1),x^{(1)}(2),...,x^{(1)}(15)\}$$

$$x^{(1)}(k)=\sum_{i=1}^{k}x^{(0)}(i)(k=1,2,3,...15)$$

$$z^{(1)}(k)=0.5x^{(1)}(k)+0.5x^{(1)}(k-1)(k=2,3,4,...15)$$

5.2.3 预测系数

$$u=(a,b)^T$$

$$Y=\{x^{(2)}(2),x^{(2)}(3),...,x^{(2)}(15)\}^T$$

$$B=\begin{bmatrix}-z^{(1)}(2) & 1\\ -z^{(1)}(3) & 1\\ ..... & \\ -z^{(1)}(15) & 1\end{bmatrix}$$

$$u = (a,b)^T = (B^T B)^{-1} B^T Y$$

5.2.4 生成数预测函数

$$x^{(1)}(k+1) = (x^{(0)}(1) - \frac{b}{a})e^{-ak} + \frac{b}{a}$$

5.2.4 得到预测数据

由于我们得到的预测函数是“生成数”的预测函数，所以我们需要用生成数来还原原始数据，从而得到对原始数据的预测。

$$x^{(0)}(k+1) = x^{(1)}(k+1) - x^{(1)}(k)$$

$$x^{(0)}(16) = x^{(1)}(16) - x^{(1)}(15)$$

5.3 Map-Reduce 模型

该任务与任务 3 极为类似，主要有五处不同：

5.3.1 不合理数据过滤

该任务不仅对数据格式不合理的文本行进行异常捕获并过滤，而且要求时间必须在“2015-09-08~2015-09-22”这 15 天之中，如果日期在其它时间中则将其过滤掉，因为这会影响后面的预测算法。

5.3.2 排序

排序先按小时排序，再按接口二次排序，再按天数三次排序。

5.3.3 分组

分组按小时时间窗分组（注意这里的小时不考虑天数的不同，仅考虑小时），统计每一个接口在每一个时间窗中“15 天的访问频次”，从而得到 15 个离散数据。

5.3.4 Reduce 端

Reduce 中，在每一次 reduce()方法的调用中都是处理一组数据，在该组数据中，小时一样，接口和天数不同，但是这些数据是按照“接口>>>天数”的次序进行排序的，所以我们很容易可以得到对于任意一个小时时间窗，任意一个接口在 15 天中每天的访问频次，从而得到 15 个数据。

5.3.5 灰色预测

Reduce 中对于得到的 15 个离散数据调用“灰色预测模型”算法得到第 16 天的预测值。

5.4 优化

5.4.1 自适应滤波法的缺点

最初选取自适应滤波法作为预测算法，考虑该算法的技术实现比较简单，且利用了全部历史数据通过不断地自我学习来寻求“使得误差可以忍受”的最佳权向量，在学习过程中不断利用现有误差和原始数据来修正权值。

预测算法为：

$$\hat{y}_{t+1} = w_1 y_t + w_2 y_{t-1} + \ldots + w_N y_{t-N+1}$$

权向量调整算法为：

$$w_i^{'} = w_i + 2ke_{i+1}y_{t-i+1}$$

但是自适应滤波法有其缺点：

- 初始值难以确定。学习常数 k，权向量的初始值，权数的个数 N 都难以确定，且不同的取值对最终预测结果影响大

- 在某些数据下无法收敛
- 不能预测趋势型的数据序列

5.4.2 改进为灰色预测模型

- 灰色预测模型适应原始数据比较少的情况
- 能够充分利用生成数来发掘数据中潜在的规律
- 不存在无法收敛的情况

  利用灰色预测模型很好地对数据进行了预测。