

课程设计1 – 日志统计分析 实验报告

陈越琦

(121160005 Yueqichen.0x0@gmail.com)

刘威

(131220085 liuwei13cs@smail.nju.edu.cn)

杨杰才

(131220115 mark_grove@qq.com)

周子博

(121250229 441842096@qq.com)

摘要：本次实验是本课程的最后的综合实验。在本次实验中，我们小组通过使用MapReduce编程框架实现日志分析，了解、掌握并使用了以下两点MapReduce 编程技能：1、海量日志数据的统计分析 2、基于MapReduce 的预测模型设计：通过对历史日志数据的分析建立预测模型 3、以及拓展部分：IP-接口对应关系分析。

目录

§1. 引言	3
§2. 实验环境与概述	3
§3. 实验所用算法与设计流程	4
3.1 统计部分	4
3.1.1 总体设计	4
3.1.2 任务 1	4
3.1.3 任务 2-4	7
3.2 预测部分	9
3.2.1 Step1 概述	10
3.2.2 Step1-1: 平均值	11
3.2.3 Step1-2: 插值	14
3.2.4 Step1-3: 线性回归	17
3.2.5 Step2: 计算RMSE	20
3.3 拓展分析部分	22
§4. 实验编译运行与结果展示分析	23
4.1 源代码编译说明	23
4.2 JAR包执行方式说明	23
4.3 实验结果截图与分析	24
§5. 实验优化与性能分析	36
5.1 实验优化与优化效果展示	36
5.1.1 通用优化手段	36
5.1.2 统计型任务的优化	37
5.1.3 预测型任务的优化	37
5.2 实验性能分析	39
§6. 实验总结	40
6.1 实验内容总结	40
6.2 团队合作总结	40
6.3 实验任务分工	41
§7. 附录: 各任务执行报告截图	42
7.1 统计型任务	42
7.2 预测型任务	46

§1. 引言

本次实验，我们小组通过使用 MapReduce 来实现日志分析。

根据实验要求我们可以将任务分成统计型和预测型两类。

任务类型	具体任务
统计型	状态码出现频次总计和分时间窗（小时）统计 IP访问频次总计和分时间窗（小时）统计 URL访问频次统计和分时间窗（秒）统计 URL响应时间分时间窗（小时）统计
预测型	URL访问频次预测（每个小时每个接口）

由上面的分析结合实际的实验过程中的一些想法与探索，在本次实验中，我们主要完成了以下几个方面的实验内容：1. 对日志数据进行统计分析；2. 使用历史数据对接口访问频次进行预测；3. 对日志分析进行一定扩展，从所给日志中分析一些有用信息。日志分析在互联网企业应用很广，通过完成本次课程设计，我们也进一步了解MapReduce 技术在工业界的应用。

我们将在第2部分描述实验环境，在第3部分详细解释程序设计的主要流程以及程序采用的主要算法，在第4部分中展示结果，优化和性能分析放在第5部分，第6部分总结报告。附录为执行报告。

§2. 实验环境与概述

本次实验的本地开发与测试环境如下：

类别	版本号
Linux版本	Ubuntu-15.04
JDK版本	7u65
Hadoop版本	2.7.1

图 1 开发与测试环境

对于各个任务，我们使用随机抽取生成的小数据集来调试程序的正确性，再用大数据集进行性能调优与最终结果测试。在RMSE计算部分，我们在初始时没有考虑验证计算结果的正确性，导致后面对方法好坏比较部分走了一定的弯路，经过修改后才获得了正确的结果。

§3. 实验所用算法与设计流程

3.1 统计部分

本部分对应实验要求中的任务1 - 任务4。本部分所用算法基本来自于课本专利文献分析算法部分，本部分采用的主要算法与技术有：MapReduce用于计数和统计，复合键的使用，自定义的 Partitioner 与 Combiner，多文件输出与文件名的定义。

3.1.1 总体设计

四个统计型任务的设计方法大体是一致的。根据具体任务需求得到相应的信息单元(information unit)和对应的时间粒度(time)。Table1是其对应关系。

具体任务	信息单元(IU)	时间粒度(Time)
状态码出现频次总计和分时间窗(小时)统计	状态码	小时
IP访问频次总计和分时间窗(小时)统计	IP	小时
URL访问频次统计和分时间窗(秒)统计	URL	秒
URL响应时间分时间窗(小时)统计	响应时间	小时

表 1 具体任务与信息单元，时间粒度对应关系

对于信息单元的切割问题，通过查看日志文件我们可以看到：对日志文件以空格进行切分得到的字符串子串的数目是不一致的。因而在具体的切割时要考虑到这一点，才能得到正确的信息单元。我们在初始时，没有考虑全部情况，导致在状态码统计时出现了较大的偏差。

将这四个任务根据 Reduce 操作的不同进行进一步细分，可分为两部分(任务1与任务2-4)：对第一个任务，所有的结果要全部输出在一个文件中，因而需要关注的重点（即最终输出的键）分别为信息单元和时间。前一部分，我们关注的是每个信息单元的统计结果；而在后一部分，我们关注的是每个时间窗对应的各个信息单元的统计信息。由上分析，我们在Reduce部分采用了wordcount + 倒排索引的设计方法。而对后三个任务，每个信息单元对应一个输出文件，我们只需关注信息单元及其统计信息即可，因而不需要使用特殊的设计。因而，对于前四个任务，我将分为 任务1 与 任务2-4 两部分分别进行深入阐述。

3.1.2 任务 1

由上面的分析，我们在任务1中采用了wordcount + 倒排索引的思路来完成整个设计过程。使用 **wordcount** 的思路统计日志中各个状态码(200, 404, 500)出现总的频次，使用倒排索引的思路按照小时时间窗输出各个时间段各状态码的统计情况。

任务1的具体MapReduce 设计，Map/Reduce键值对与伪代码如下：

考虑到要求先输出总体的统计结果再输出具体的分时间窗口的统计信息，因此设计键值对转换过程如下(Figure 2)；

	输入键	输入值	输出键	输出值
Map	记录 offset	一个日志记录	0#信息单元 1#时间粒度#信息单元	1 1
Combine	0#信息单元 1#时间粒度#信息单元	1 1	0#信息单元 1#时间粒度#信息单元	频次汇总 频次汇总
Partition	按信息单元做分区			
Reduce	0#信息单元 1#时间粒度#信息单元	频次汇总 频次汇总	信息单元 时间粒度	频次汇总 信息单元： 频次汇总

图 2 任务1 键值对转换过程

可以看到，在Map阶段，我们输出了两种键，一种以”0#”开头，一种以”1#”开头。它们分别对应了统计各个状态码出现的总频次与按照小时时间窗输出各个时间段各状态码的统计结果（即 wordcount 与倒排索引部分）。由于在Reduce之前键值对会按照key进行排序，这样便保证了输出结果与顺序的正确性。而在Reduce阶段，通过判断键的第一部分，便可以确定进行何种处理（wordcount 或倒排索引）。

同时，使用这种方法也可以确保在按照时间窗进行统计时，最终的输出结果是正确的，并且按照时间递增的顺序显示。

具体的 Map 与 Reduce 各阶段的设计如下：

1. **Map** 读入每条日志记录，切割记录得到相应粒度的时间和信息单元，输出“0#信息单元”方便在Reduce阶段进行总的统计，输出“1#时间粒度#信息单元”方便在Reducer阶段得到相应时间窗口中的统计结果。
2. **Combine** 对有相同键的Map输出，将其频次相加汇总，较少中间数据传输。
3. **Partition** 根据信息单元进行分区，将拥有不同的IP，URL，或者状态码记录分到不同的Reduce节点。
4. **Reduce** 因为MapReduce框架会对key进行排序，所以所有”0”开始的键值对在所有”1”开始的键值对的前面。因此可以在所有”0”开始的键值对上进行总的统计工作，从而输出总次数。在”1”开始的键值对上进行倒排索引统计，统计过程类似于实验二，代码也由实验二相应的 reducer 部分修改而来。

相对应的各部分的伪代码如下：

Mapper:

Class 1 *class Mapper*

Input : 日志记录**Output :** 键值对 (0#信息单元, 1#时间粒度#信息单元)

```

1: procedure MAP(Object key, Text line)
2:   Logs[] (切割出的子串组成数组)  $\leftarrow$  line
3:   IU(信息单元),time(时间)  $\leftarrow$  Logs[]
4:   Emit(0#IU,one)
5:   Emit(1#time#IU,one)
6: end procedure

```

Combiner:

Class 2 *class Combiner*

Input : 键值对 (0/1#信息单元, 出现次数)**Output :** 键值对 (0/1#时间#信息单元, 出现次数)

```

1: procedure COMBINE(Text key, Iterable<IntWritable> value)
2:   Sum = 0
3:   for all val in value do do
4:     Sum  $\leftarrow$  Sum + val
5:   end for
6:   Emit(key,Sum)
7: end procedure

```

NewPartitioner:

Class 3 *class NewPartitioner*

```

1: procedure GETPARTITION(Text key, IntWritable value, int NumReduceTasks)
2:   term  $\leftarrow$  key.split("#")[1]
3:   return super.getPartition(term, value, NumReduceTasks)
4: end procedure

```

Reducer:

Class 4 class Reducer

Input : Combiner的结果<Text key, Iterable<IntWritable> >**Output :** 1. state(状态码):总频次**Output :** 2. 时间窗 state(状态码):总频次

```

1: procedure SETUP( )
2:    $t_{prev} \leftarrow \emptyset$ 
3:    $P \leftarrow newPostingsList$ 
4: end procedure
5: procedure REDUCE(Text key(tuple< 0/1, t, n >), IntWritable values)
6:   if key.startwith("0") then
7:      $0 \leftarrow sum$ 
8:     for val in values do
9:        $sum \leftarrow sum + val$ 
10:    end for
11:    Emit(t:sum);
12:   end if
13:   if key.startwith("1") then
14:      $0 \leftarrow sum$ 
15:     for val in values do
16:        $sum \leftarrow sum + val$ 
17:     end for
18:     if  $t \neq t_{prev}$  &&  $t_{prev} \neq \emptyset$  then
19:       Emit( $t_{prev}, P$ )
20:        $P.Reset()$ 
21:        $P.Add(< n, sum >)$ 
22:      $t_{prev} \leftarrow t$ 
23:   end if
24: end procedure
25: procedure CLEANUP( )
26:   Emit( $t, P$ )
27: end procedure

```

3.1.3 任务 2-4

任务2-4 本质上大体相同，均是进行较为简单的求和处理。在这三个任务中，最关键的一点便是按照要求进行多文件输出。在多文件输出部分，我使用了MultipleOutputs 类来实现根据信息单元的不同，将结果输出到相应的文件中。MultipleOutputs 的具体使用方法如下

图：

```

protected void setup(Context context) throws IOException, InterruptedException
{
    out = new MultipleOutputs<Text,Text>(context);
}

        out.write("IPResult",result, new Text(""),IP+".txt");

protected void cleanup(Context context) throws IOException, InterruptedException
{
    out.close();
}

```

图 3 MulitpleOutputs 示例

由上所示，通过 MultipleOutputs 类的 write 函数，我们便可以根据信息单元来将内容输出到合适的文件之中。

由前面的分析可知，任务2-4与任务1中各个部分键值对的类型与内容构成完全一致（见图2）。

具体的 Map 与 Reduce 各阶段的设计如下：

1. **Map** 读入每条日志记录，切割记录得到相应粒度的时间和信息单元，输出“0#信息单元”方便在Reduce阶段进行总的统计，输出“1#时间粒度#信息单元”方便在Reducer阶段得到相应时间窗口中的统计结果。
2. **Combine** 对有相同键的Map输出，将其频次相加汇总，较少中间数据传输。
3. **Partition** 根据信息单元进行分区，将拥有不同的IP，URL，或者状态码记录分到不同的Reduce节点。
4. **Reduce** 因为MapReduce框架会对key进行排序，所以所有”0”开始的键值对在所有”1”开始的键值对的前面。因此可以在所有”0”开始的键值对上进行总的统计工作，在”1”开始的键值对进行各个时间窗上的统计，并根据信息单元的内容输出到相应的文件之中。在任务4中，总次数代表总的响应时间，最终输出时除以访问次数，得到平均响应时间。

任务2-4的 Map, Combine 与 Partition 部分与任务1完全一致，主要的差别体现在Reduce部分，因而此处不再重复给出伪代码。下面是任务2-4 的的Reduce 伪代码(括号内是 任务4 添加的部分)：

Reducer:**Class 5 class Reducer****Input :** Combiner的结果<Text key, Iterable<IntWritable> >**Output :** 1. IU(信息单元):总频次**Output :** 2. 时间窗 IU(信息单元):总频次

```

1: procedure SETUP( )
2:     out ← MultipleOutputs(context)
3: end procedure
4: procedure REDUCE(Text key(tuple<0/1,t,n>), IntWritable values)
5:     if key.startwith("0") then
6:         0 ← sum (0 ← num)
7:         for val in values do
8:             sum ← sum + val (num ← num + 1)
9:         end for
10:        out.write(t:sum/(t:sum/num),t+.txt");
11:    end if
12:    if key.startwith("1") then
13:        0 ← sum
14:        for val in values do
15:            sum ← sum + val
16:        end for
17:        out.write(t,n:sum/(n:sum/num),n+.txt");
18:    end if
19: end procedure
20: procedure CLEANUP( )
21:     out.close()
22: end procedure

```

3.2 预测部分

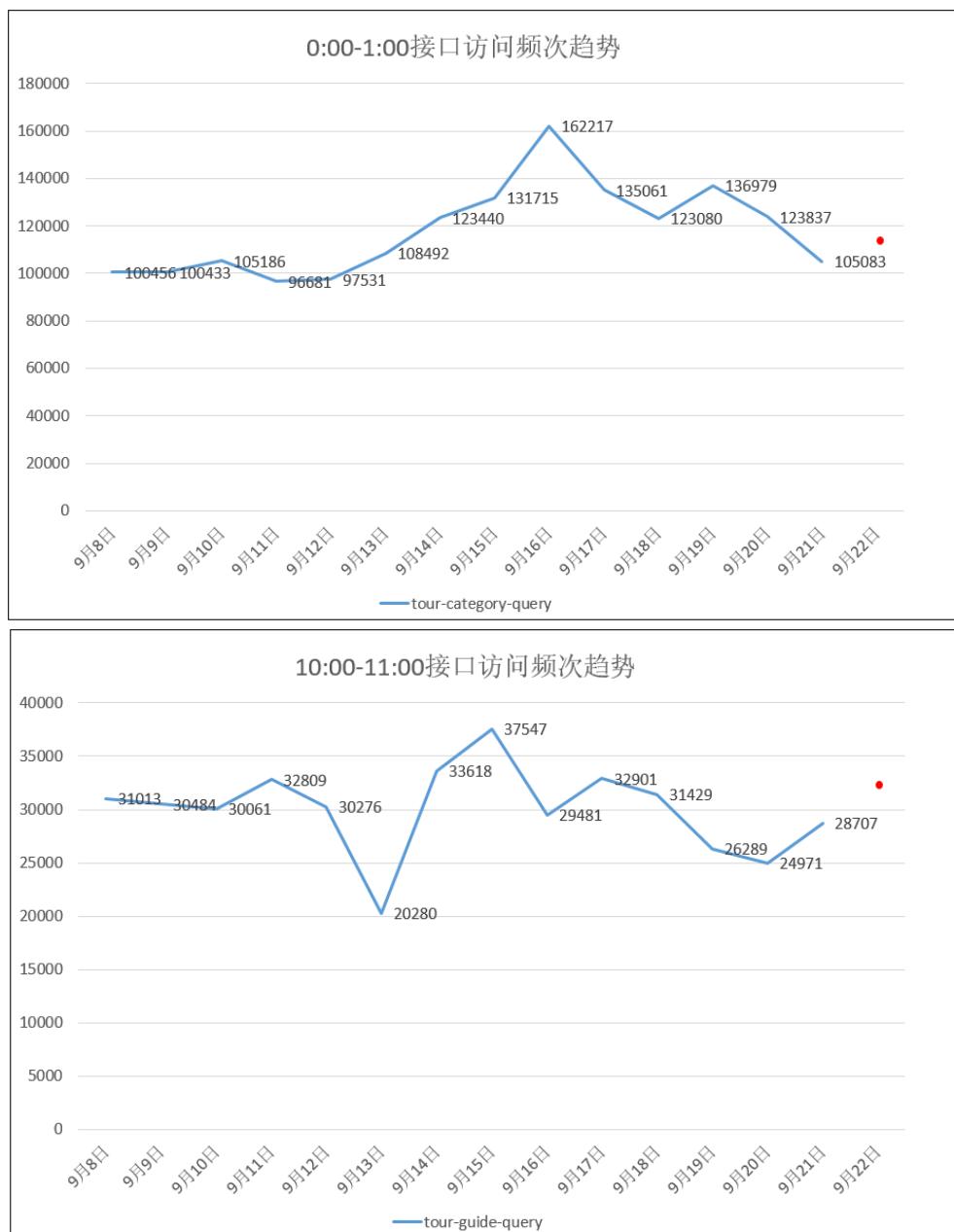
本部分总体上可分为两个部分：

- Step1: 预测接口访问次数并输出预测结果和真实结果。
- Step2: 从Step1的输出结果中计算RMSE。

下面我们将分别叙述 Step1 与 Step2 所采用的算法与设计流程：

3.2.1 Step1 概述

如果要使用22号之前各个url在不同时间窗的访问量来预测22各个url在相应时间窗的访问量，主要是通过观察22号之前的访问数据的情况是否具有某种规律——递增，递减，周期性或者其他规律。因此我们对22号之前的访问情况进行了统计，并统计出各时间窗下接口访问量变化的情况。（为了提高预测的精度，对于部分存在问题的记录与日期出现错误的记录，我们直接将其忽略。）下面是几个时间窗下示例接口的随时间变化而访问次数变化的图像(图中红点是22日访问次数的真实值)：



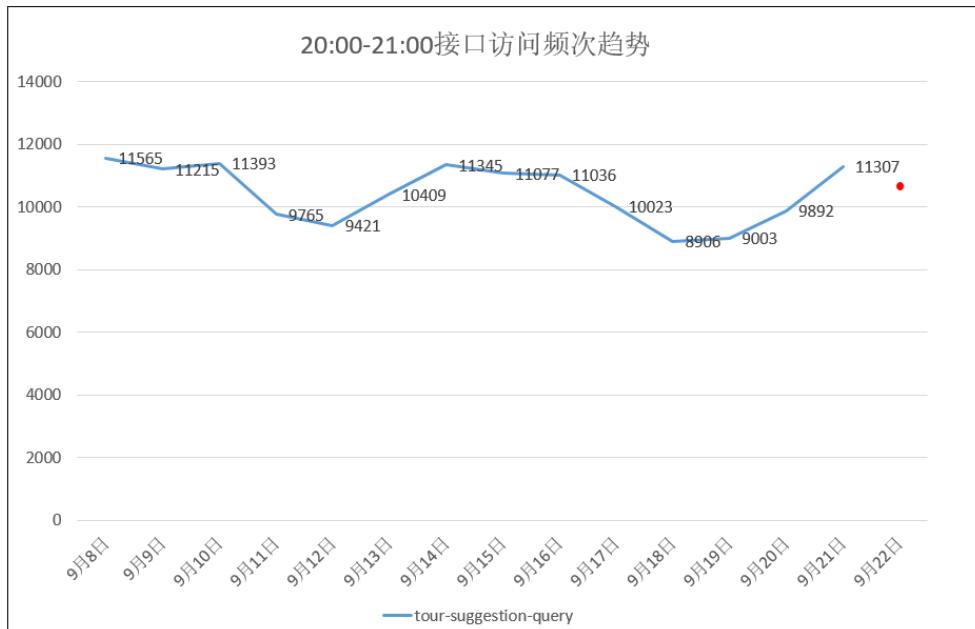


图 4 接口访问次数变化曲线图示例

通过对上图所示及其他接口的访问次数的统计与变化结果进行分析，我们发现数据中并不存在简单明显的规律，因此我们根据一些接口的变化情况，确定了三种基本的预测手段：(1) 使用访问次数的平均值进行预测；(2) 使用代数多项式插值——拉格朗日插值方法进行预测；(3) 使用线性回归进行预测。在后面我们将分别叙述三种方法的原理与实现，并在后面的章节比较他们的效果，确定最终采用的方法。

3.2.2 Step1-1: 平均值

面对这样的预测问题，我们首先想到的方法便是以前14天对应时间窗的访问次数的平均值来作为22日访问次数的预测值。这是最简单的方法，也可以作为其他方法比较的baseline。将其他方法与此方法所得 RMSE 值进行比较，可以更好地体现预测结果的好坏。

本方法的基本思路便是将每个接口在每日每个时间窗的访问次数进行统计，求出每个接口在每个时间窗的前14天的访问次数的平均值并将其与22日的真实值一起输出，作为计算 RMSE 值的输入。

本部分的MapReduce 设计如下：

键值对的转换过程如下图：

	输入键	输入值	输出键	输出值
Map	记录 offset	一个日志记录	url#小时#日期	1
Combine	url#小时#日期	1	url#小时#日期	频次汇总
Partition	按 url 划分			
Reduce	url#小时#日期	频次汇总	url#小时	预测值#真实值

图 5 本部分的键值对转换

由上面的叙述，我们可以得到本部分如下的伪代码设计：

Mapper:

Class 6 class Mapper

Input : 日志记录

Output : 键值对 (url#小时#日期)

```

1: procedure MAP(Object key, Text line)
2:   if 记录为完整并且正常的记录 then //如果记录有效，即记录完整并且日志一致
3:     url = getUrl(line); //切割得到URL
4:     hour = getHour(line); //切割得到小时为单位的时间窗
5:     date = getDate(line); //切割得到记录日期
6:     Emit(url#hour#date, 1);
7:   end if
8: end procedure

```

Combiner:

Class 7 class Combiner

Input : 键值对 (url#小时#日期, 1)

Output : 键值对 (url#小时#日期, 频率汇总)

```

1: procedure COMBINE(Text key, Iterable<IntWritable> value)
2:   Sum = 0
3:   for all val in value do do
4:     Sum ← Sum + val
5:   end for
6:   Emit(key, Sum)
7: end procedure

```

NewPartitioner:

Class 8 class NewPartitioner

```
1: procedure GETPARTITION(Text key, IntWritable value, int NumReduceTasks)
2:     term ← key.split("#")[0]
3:     return super.getPartition(term, value, NumReduceTasks) //根据URL划分数据
4: end procedure
```

Reducer:

Class 9 class Reducer

Input : Combiner的结果<Text key, Iterable<IntWritable> >**Output :** [url#小时#日期, 预测值#真实值]

```

1: procedure SETUP( )
2:    $t_{prev} \leftarrow \emptyset$ 
3:    $P \leftarrow newPostingsList$ 
4: end procedure
5: procedure REDUCE(key, value)
6:   if thent  $\neq t_{prev}$  &&  $t_{prev} \neq \emptyset$  then
7:     sum = 0;
8:     for each record r in P do
9:       sum += r.getValue();
10:    end for
11:    pred = sum/14;
12:    Emit( $t_{prev}, pred\#realValue$ )
13:    P.Reset()
14:  end if
15:  if getHour(key) == 22 then
16:    realValue = value; //获得真实值
17:  else
18:    P.Add(<n, f>); // 记录22号之前的访问频率
19:  end if
20:   $t_{prev} \leftarrow getUrl(key) + "\#" + getHour(key)$ 
21: end procedure
22: procedure CLOSE( )
23:   sum = 0;
24:   for each record r in P do
25:     sum += r.getValue();
26:   end for
27:   pred = sum/14;
28:   Emit( $t_{prev}, pred\#realValue$ )
29: end procedure

```

3.2.3 Step1-2: 插值

在前面的基础上，我们假设存在一个多项式函数表示了各个接口访问次数随时间变化而变化的情况。而确定这一函数在某点取值的一种重要方法便是插值法。插值法是利用已有的

训练数据作为 $f(x)$ 在某区间中已知的若干点的函数值，作出适当的特定函数，在区间的其他点上用这特定函数的值作为函数 $f(x)$ 的近似值的方法。使用这种方法便可以实现对未知数据的预测。因而，我们使用了多项式插值中常用的拉格朗日插值法来进行预测。

拉格朗日插值法基本原理如下：假设有 $n + 1$ 个互异点 $(x_i, y_i), i = 0, \dots, n$ ，构造拉格朗日插值多项式为

$$L_n(x) = y_0 l_0(x) + y_1 l_1(x) + \dots + y_n l_n(x) = \sum_{i=0}^n y_i l_i(x) \quad (1)$$

其中 n 次多项式

$$l_i(x) = \frac{(x - x_0) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)} \quad (2)$$

为拉格朗日基函数。

根据插值多项式唯一性，拉格朗日插值多项式即为所求插值函数。对于插值余项，即截断误差 $R_n(x) = f(x) - L(x)$ ，有拉格朗日余项定理如下：设 $f(x)$ 在区间 $[a, b]$ 上存在 $n + 1$ 阶导数， $x_i \in [a, b] (i = 0, 1, \dots, n)$ 为 $n + 1$ 个互异节点，则对任何 $x \in [a, b]$ 有

$$R_n(x) = f(x) - L_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} w_{n+1}(x) \quad (3)$$

其中 $w_{n+1}(x) = \prod_{i=0}^n (x - x_i)$, $\xi \in (a, b)$

一般来说外推的效果要比内插的效果差，此外并不是插值多项式的次数越高，插值效果越好，精度也不一定是随着次数的提高而升高，这种现象在上个世纪由Runge发现，称为Runge现象。一般使用分段插值的方法缓解。

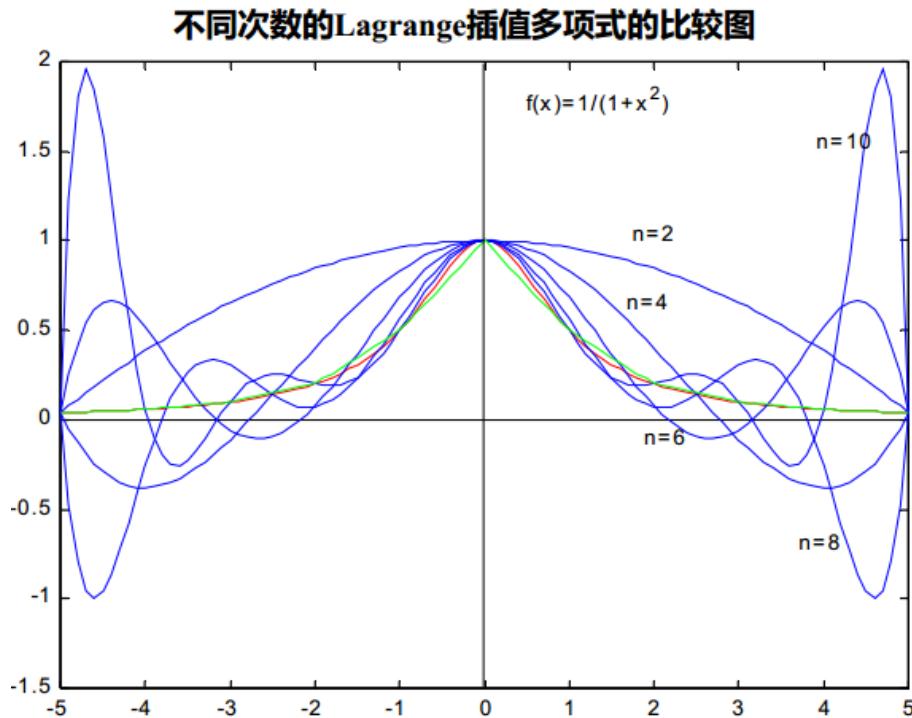


图 6 Runge现象

在本任务中，要求预测22号的url访问情况，因此只要使用22号前几天的数据即可，而不需要使用完整的14天的数据。

使用该算法的具体MapReduce设计如下：

本部分的键值对转换如下：

	输入键	输入值	输出键	输出值
Map	记录 offset	一个日志记录	url#小时#日期	1
Combine	url#小时#日期	1	url#小时#日期	频次汇总
Partition	按 url 划分			
Reduce	url#小时#日期	频次汇总	url#小时	预测值#真实值

图 7 插值过称中键值对转换

值得注意的是，Map阶段的输出键为“url#小时#日期”组合，因为在Reduce之前会对分配到同一个Reducer节点的键进行排序，因此能够保证具有相同url和相同时间窗的键值对被分配到一起，并且按照日期从前到后排序（事实上插值方法只要求互异而不要求有序）。

本方法出 Reducer 外的设计与 Step 1-1 完全一致，因而我们在这只列出新的 Reducer 的伪代码：

Reducer:

Class 10 class Reducer

Input : Combiner的结果<Text key, Iterable<IntWritable> >

Output : [url#小时#日期, 预测值#真实值]

```

1: procedure SETUP( )
2:    $t_{prev} \leftarrow \emptyset$ 
3:    $P \leftarrow newPostingsList$ 
4: end procedure
5: procedure REDUCE(key, value)
6:   if  $t \neq t_{prev}$  &&  $t_{prev} \neq \emptyset$  then
7:     利用P中记录根据式1,2计算得到预测值pred
8:     Emit( $t_{prev}, pred \# real - value$ )
9:      $P.Reset()$ 
10:    if getHour(key) == 22 then
11:      real-value = value; //获得真实值
12:    else
13:       $P.Add(< n, f >);$  // 记录22号之前的访问频率
14:    end if
15:     $t_{prev} \leftarrow getUrl(key) + \# + getHour(key)$ 
16: end procedure
17: procedure CLOSE( )
18:   利用P中记录根据式1,2计算得到预测值pred
19:   Emit( $t_{prev}, pred \# true$ )
20: end procedure

```

3.2.4 Step1-3: 线性回归

而对于预测连续值得任务，使用机器学习中的回归算法也是一个经常被使用的方法。因而，在本部分，我们还尝试了使用线性回归来进行预测。

线性回归，是利用数理统计中回归分析，来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法，运用十分广泛。在线性回归模型中，最简单的是一元一次线性模型。对该模型进行扩充主要有两种方向。一种方向是多元模型，一种方向是多次模型。多元适用于多个参数条件的情况.而由于受到前面插值所得结果的影响，对于本任务，较复杂的模

型不一定能获得更好的效果。因而，我们最终还是选择较为简单的一次线性回归模型。

此时，该模型可以表示为： $y = x_0 + x_1 * t$ 。要求解这个模型以进行预测，即是使用已有的数据求出 x_0 与 x_1 的值，并使用新的 t 值来进行预测。而求解模型的最常用的方法便是最小二乘法：最小二乘法是一种数学优化技术。它通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和为最小。

使用最小二乘法来求解这一模型即是求解如下的优化问题：

$$\min_{x_0, x_1} \left\| \begin{pmatrix} 1 & t_1 \\ \vdots & \vdots \\ 1 & t_n \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \right\|_2 = \min_x \|Ax - b\|_2$$

进行微分求最值可以得到该问题的闭式解如下：

$$x_1 = \frac{\sum_{i=1}^n (t_i - \frac{1}{n} \sum_{j=1}^n t_j)(y_i - \frac{1}{n} \sum_{j=1}^n y_j)}{\sum_{i=1}^n (t_i - \frac{1}{n} \sum_{j=1}^n t_j)^2} \quad (4)$$

$$x_0 = \frac{1}{n} \sum_{i=1}^n y_i - x_1 \frac{1}{n} \sum_{i=1}^n t_i \quad (5)$$

通过上面的解，我们便可以得到使用训练数据来进行模型参数的求解方法，从而进行预测。因而，我们可以得到如下的MapReduce设计方式：首先，统计出各个接口在各个时间窗内15天每天的访问次数，通过闭式解计算参数模型并预测第十五天的访问频次，将其与真实值组合，作为复合键值一起输出。

本部分如下的伪代码如下：

Mapper:

Class 11 class Mapper

Input : 日志记录

Output : 键值对 (url#小时每天的访问次数统计)

```

1: procedure MAP(Object key, Text line)
2:   if 记录为完整并且正常的记录 then //如果记录有效，即记录完整并且日志一致
3:     初始化数组In[15] 并初始化为0
4:     url = getUrl(line); //切割得到URL
5:     hour = getHour(line); //切割得到小时为单位的时间窗
6:     date = getDate(line); //切割得到记录日期
7:     In[date] = 1;
8:     Emit(url#hour, In[0]+In[1]+...+In[14]);
9:   end if
10: end procedure

```

Combiner:

Class 12 class Combiner

Input : 键值对 (url#小时每天的访问次数统计)

Output : 键值对 (url#小时每天的访问次数统计)

```

1: procedure COMBINE(Text key, Iterable<Text> value)
2:   初始化数组In[15] 并初始化为0
3:   for all val in value do do
4:     Ins[] ← val进行切割//Ins数组存储键中所存储的每天的访问次数统计
5:     for i = 0...14 do
6:       In[i] ← In[i] + Ins[i]
7:     end for
8:   end for
9:   Emit(url#hour, In[0] + In[1] + ... + In[14])
10: end procedure

```

NewPartitioner:

Class 13 class NewPartitioner

```

1: procedure GETPARTITION(Text key, Text value, int NumReduceTasks)
2:   term  $\leftarrow$  key.split("#")[0]
3:   return super.getPartition(term, value, NumReduceTasks) //根据URL划分数据
4: end procedure

```

Reducer:

Class 14 class Reducer

Input : Combiner的结果<Text key, Iterable<Text>>**Output :** [url#小时, 真实值#预测值]

```

1: procedure REDUCE(key, value)
2:   初始化数组In[15] 并初始化为0
3:   for all val in value do do
4:     Ins[]  $\leftarrow$  val进行切割//Ins数组存储键中所存储的每天的访问次数统计
5:     for i = 0...14 do
6:       In[i]  $\leftarrow$  In[i] + Ins[i]
7:     end for
8:   end for
9:   根据前面的闭式解, 通过每天的统计数据计算出平均值, x0 与 x1 的值
10:  res  $\leftarrow$  In[13] + x1; // 获得预测值
11:  Emit(url#hour,真实值#预测值)
12: end procedure

```

3.2.5 Step2: 计算RMSE

RMSE的计算可以分为计算差值与求和开方两个部分。在计算过程中, 前者由 Map 过程完成, 计算每个预测值与真实值的差值的平方; 后者由 Reduce 过程来完成, 进行求和, 开方以及除法运算得到最终的RMSE的计算结果。

本部分的具体 MapReduce 设计流程如下:

RMSE计算部分的键值对转换为

	输入键	输入值	输出键	输出值
Map	插值结果 offset	一条插值结果	小时	(预测值 - 真实值) ²
Reduce	小时	(预测值 - 真实值) ²	RMSE	

图 8 RMSE计算过称中键值对转换

本任务各个过程的伪代码如下：

Mapper:

Class 15 class Mapper

Input : 插值结果

Output : 键值对 (url#小时#日期, (预测值-真实值)²)

```

1: procedure MAP(Object key, Text value)
2:   hour = getHour(value); //获得所属小时
3:   pred = getPred(value); //得到预测值
4:   real = getReal(value); //得到真实值
5:   Emit(hour, (pred - real)2);
6: end procedure

```

Reducer:

Class 16 class Reducer

Input : Mapper的结果<Text key, Iterable<LongWritable> values >

Output : RMSE

```

1: procedure SETUP( )
2:   M = 0
3:   total = 0
4: end procedure
5: procedure REDUCE(key, Iterable < LongWritable > values )
6:   M = M + 1;
7:   sum = 0;
8:   N = 0;
9:   for each vl in values do //对属于同一个小时窗口的(预测值-准确值)2
10:    sum += vl;
11:    N = N + 1;
12:   end for
13:   total += sqrt (sum / N); //将所有小时窗口的计算结果相加
14: end procedure
15: procedure CLOSE( )
16:   Emit(total / M); //最后计算出RMSE并输出
17: end procedure

```

3.3 拓展分析部分

在拓展部分，我们受到了课桌上单词共现的启发，考虑到对某IP与某接口共同出现的次数进行统计，从而确定出某个IP在某段时间内对各个接口的访问频次。通过这种统计过程，我们可以分析出每个IP对于接口的访问的偏好，从而可以进行相应的访问优化，比如将这些接口存储在相同的server上，减少路由选择的时间。同时，这种数据也可以对IP的异常访问进行判断（例如DDos攻击）并及时作出处理。具体的分析可以查看下一章节。

本部分的实现较为简单，思想上与其他普通的统计任务类似。具体的MapReduce 设计为：读取每一条记录，切割出IP与访问接口，将其作为复合键，统计每个键的输出次数再输出。最终的输出结果进行进一步的分析即可得到更多信息。

本部分的具体伪代码如下：

Mapper:

Class 17 class Mapper

Input : 日志记录

Output : 键值对 (IP#URL 访问频次)

```

1: procedure MAP(Object key, Text line)
2:   Logs[] (切割出的子串组成数组) ← line
3:   IP, URL ← Logs[]
4:   Emit(IP#URL, one)
5: end procedure

```

Combiner:

Class 18 class Combiner

Input : 键值对 (IP#URL 访问频次)

Output : 键值对 (IP#URL 访问频次)

```

1: procedure COMBINE(Text key, Iterable<IntWritable> value)
2:   Sum = 0
3:   for all val in value do do
4:     Sum ← Sum + val
5:   end for
6:   Emit(key, Sum)
7: end procedure

```

Reducer:

Class 19 class Reducer

Input :键值对 (IP#URL 访问频次)**Output :**键值对 (IP#URL 访问频次)

```

1: procedure REDUCE(Text key, Iterable<IntWritable> value)
2:   Sum = 0
3:   for all val in value do do
4:     Sum ← Sum + val
5:   end for
6:   Emit(key, Sum)
7: end procedure

```

可以看到，这个任务就是一个较为简单的统计过程。但其中蕴含了很多信息，对于这部分的分析，请查看4.3节。

§4. 实验编译运行与结果展示分析

4.1 源代码编译说明

本次实验的源代码主要分为三个部分(分别对应第三章下的三个子部分):

- Stat
- Pred
- Ana

每一部分各有一个Driver 将各个job组合在一起。编译时将各个部分对应的文件夹下与其子文件夹下所有的.java文件同时进行编译，然后将得到的所有.class文件置于同一目录之下打包得到最终的jar包 (打包时指定主类为 Driver)。

4.2 JAR包执行方式说明

1. 对Stat 部分按照如上所述的方式编译得到jar包，并上传至集群。本jar包对应任务1 - 任务4。
2. 使用hadoop jar Stat.jar /data/task1/JN1.LOG/2015-09-08.log outputPath1 outputPath2 outputPath3 outputPath4 指令执行Stat.jar。后面四个参数分别为任务1、任务2、任务3、任务4的输出目录。
3. 对Pred 中的一个预测方式的对应部分(即Pred文件夹下的一个子文件夹)按照如上所述的方式编译得到jar包，并上传至集群。本jar包对应任务5。
4. 使用hadoop jar Pred.jar /data/task1/JN1.LOG outputPath1 outputPath2 指令执行Pred.jar。

后面两个参数分别为预测结果与预测结果与真实结果比较计算所得RMSE值的输出目录。

5. 对 Ana 下的 Ana1 文件夹的对应部分按照如上所述的方式编译得到jar包，并上传至集群。本jar包对应拓展分析部分。
6. 使用hadoop jar Ana1.jar /data/task1/JN1_LOG outputPath1 指令执行Ana1.jar。第二个参数为统计结果的输出路径。

4.3 实验结果截图与分析

我们的程序均首先在本地做了简单的测试后提交到集群进行运行。(各个任务执行报告的截图见最后的附录。)

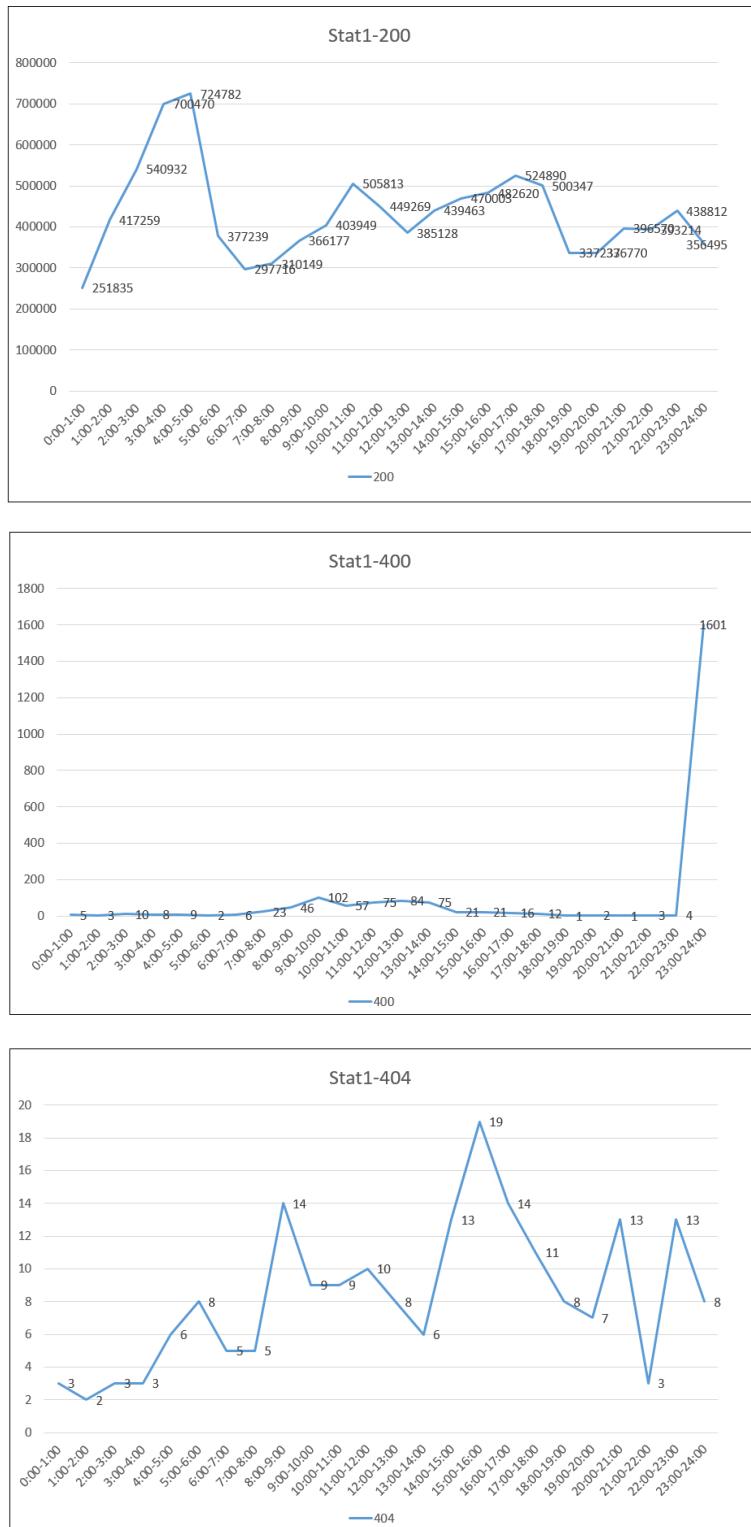
各个任务的实验结果与相应的分析如下：

1. 统计日志中各个状态码（200, 404, 500）出现总的频次，并且按照小时时间窗，输出各个时间段各状态码的统计情况。

本部分结果在HDFS上的存储路径为： hdfs://master01:9000/user/2016st21/Stat/Out1。按照要求输出结果1.txt的截图如下：

Time Window	200	404	500
0:00-1:00	200:251835	400:5	404:3
1:00-2:00	200:417259	400:3	404:2
2:00-3:00	200:540932	400:10	404:3
3:00-4:00	200:700470	400:8	404:3
4:00-5:00	200:724782	400:9	404:6
5:00-6:00	200:377239	400:2	404:8
6:00-7:00	200:297716	400:6	404:5
7:00-8:00	200:310149	400:23	404:5
8:00-9:00	200:366177	400:46	404:14
9:00-10:00	200:403949	400:102	404:9
10:00-11:00	200:505813	400:57	404:9
11:00-12:00	200:449269	400:75	404:10
12:00-13:00	200:385128	400:84	404:8
13:00-14:00	200:439463	400:75	404:6
14:00-15:00	200:470003	400:21	404:13
15:00-16:00	200:482620	400:21	404:19
16:00-17:00	200:524890	400:16	404:14
17:00-18:00	200:500347	400:12	404:11
18:00-19:00	200:337237	400:1	404:8
19:00-20:00	200:336770	400:2	404:7
20:00-21:00	200:396570	400:1	404:13
21:00-22:00	200:393214	400:3	404:3
22:00-23:00	200:438812	400:4	404:13
23:00-24:00	200:356495	400:1601	404:8

根据统计结果，我们绘制出了不同状态码在2015-09-08当天24小时内的变化情况：



由上面的图可以看出：200状态码在一整天过程中除了凌晨时间的小幅激增外，基本都处于相对平稳的状态。而在大部分时间内都没有出现太多400的状态，但在23:00-24:00这个时间

内剧增，说明可能存在DNS解析错误的问题，此时日志分析的作用就显现出来了，工程师可以根据日志反应的状态及时进行调整。同时，出现404的次数并不是很多，在20以内变化，说明服务器整体运行稳定正常。

2. 统计每个IP访问总的频次，并且按照小时时间窗，输出各个时间段各个IP访问的情况。每个IP的统计信息是一个文件，并且以IP为文件名。

本部分结果在HDFS上的存储路径为： hdfs://master01:9000/user/2016st21/Stat/Out2。按照要求，每个IP的统计信息输出到相应的txt文件中（前面为总频次，后面为分时间窗频次输出），下面我们将展示几个txt文件的截图，其余的可在HDFS或者集群用户文件夹内查看：

(1) 10.10.0.101

```
10.10.0.101:50
0:00-1:00      10.10.0.101:4
1:00-2:00      10.10.0.101:10
2:00-3:00      10.10.0.101:1
3:00-4:00      10.10.0.101:3
4:00-5:00      10.10.0.101:2
5:00-6:00      10.10.0.101:7
13:00-14:00    10.10.0.101:7
17:00-18:00    10.10.0.101:13
18:00-19:00    10.10.0.101:3
22:00-23:00    10.10.0.101:3
```

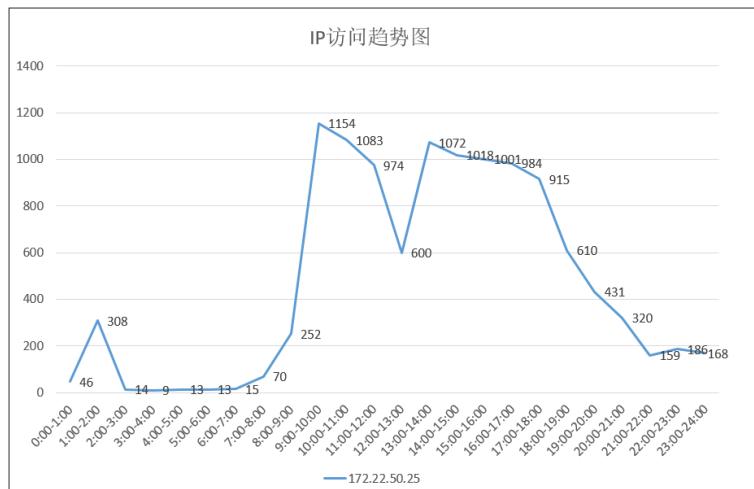
(2) 172.22.50.25

```
172.22.50.25:11415
0:00-1:00      172.22.50.25:46
1:00-2:00      172.22.50.25:308
2:00-3:00      172.22.50.25:14
3:00-4:00      172.22.50.25:9
4:00-5:00      172.22.50.25:13
5:00-6:00      172.22.50.25:13
6:00-7:00      172.22.50.25:15
7:00-8:00      172.22.50.25:70
8:00-9:00      172.22.50.25:252
9:00-10:00     172.22.50.25:1154
10:00-11:00    172.22.50.25:1083
11:00-12:00    172.22.50.25:974
12:00-13:00    172.22.50.25:600
13:00-14:00    172.22.50.25:1072
14:00-15:00    172.22.50.25:1018
15:00-16:00    172.22.50.25:1001
16:00-17:00    172.22.50.25:984
17:00-18:00    172.22.50.25:915
18:00-19:00    172.22.50.25:610
19:00-20:00    172.22.50.25:431
20:00-21:00    172.22.50.25:320
21:00-22:00    172.22.50.25:159
22:00-23:00    172.22.50.25:186
23:00-24:00    172.22.50.25:168
```

(3) 172.30.53.140



我们挑选了172.22.50.25这个IP的访问情况，按照时间窗绘制出其活动图如下：



从这里可以看出，某个IP的访问趋势图可以作为用户分类的特征，从而进一步挖掘出用户习惯和用户需求，为精准数据营销和个性化推荐奠定基础。比如类似上图IP的用户，可能更习惯在早上或者下午来访问网站、在中午时间段相对不活跃，通过分析大量的访问行为，就可以建立更加清晰的用户画像，这是目前日志分析在互联网中尤其是电商类网站的重要意义之处。

3. 统计每个接口(请求的URL)访问总的频次，并且以接口为文件，按照秒为单位的时间窗，输出各个时间段各接口的访问情况。每个接口的统计信息是一个文件，如接/tour/category/query的统计文件命名为：our-category-query.txt，每个文件的输出内容同上。

本部分结果在HDFS上的存储路径为：hdfs://master01:9000/user/2016st21/Stat/Out3。按照要求，每个接口的统计信息输出到相应的txt文件中（前面为总频次，后面为分时间窗频次输出），下面我们将展示几个txt文件的截图（由于本任务以秒为时间窗，因而输出文件较前面的任务更大），其余的可在HDFS或者集群用户文件夹内查看：

(1) tour-category-ids-query

```
□ 终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
/tour/category/ids/query:764394
00:00:00      /tour/category/ids/query:2
00:00:01      /tour/category/ids/query:9
00:00:02      /tour/category/ids/query:7
00:00:03      /tour/category/ids/query:6
00:00:04      /tour/category/ids/query:2
00:00:05      /tour/category/ids/query:7
00:00:06      /tour/category/ids/query:8
00:00:07      /tour/category/ids/query:7
00:00:08      /tour/category/ids/query:7
00:00:09      /tour/category/ids/query:5
00:00:10      /tour/category/ids/query:2
00:00:11      /tour/category/ids/query:1
00:00:12      /tour/category/ids/query:3
00:00:13      /tour/category/ids/query:8
00:00:14      /tour/category/ids/query:3
00:00:15      /tour/category/ids/query:8
00:00:16      /tour/category/ids/query:6
00:00:17      /tour/category/ids/query:3
00:00:18      /tour/category/ids/query:7
00:00:19      /tour/category/ids/query:4
00:00:20      /tour/category/ids/query:6
00:00:21      /tour/category/ids/query:3
00:00:22      /tour/category/ids/query:7
00:00:23      /tour/category/ids/query:8
00:00:24      /tour/category/ids/query:3
00:00:25      /tour/category/ids/query:3
00:00:26      /tour/category/ids/query:5
00:00:27      /tour/category/ids/query:7
00:00:28      /tour/category/ids/query:2
00:00:29      /tour/category/ids/query:3
00:00:30      /tour/category/ids/query:4
00:00:31      /tour/category/ids/query:6
00:00:32      /tour/category/ids/query:4
00:00:33      /tour/category/ids/query:3
00:00:34      /tour/category/ids/query:2
00:00:35      /tour/category/ids/query:4
00:00:36      /tour/category/ids/query:5
00:00:37      /tour/category/ids/query:7
00:00:38      /tour/category/ids/query:3
00:00:39      /tour/category/ids/query:8
00:00:40      /tour/category/ids/query:5
00:00:41      /tour/category/ids/query:7
00:00:42      /tour/category/ids/query:4
00:00:43      /tour/category/ids/query:1
00:00:44      /tour/category/ids/query:10
```

(2) tour-guide-query

```
□ 终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
/tour/guide/query:633663
00:00:00      /tour/guide/query:10
00:00:01      /tour/guide/query:3
00:00:02      /tour/guide/query:2
00:00:03      /tour/guide/query:4
00:00:04      /tour/guide/query:4
00:00:05      /tour/guide/query:10
00:00:06      /tour/guide/query:5
00:00:07      /tour/guide/query:9
00:00:08      /tour/guide/query:10
00:00:09      /tour/guide/query:3
00:00:10      /tour/guide/query:4
00:00:11      /tour/guide/query:11
00:00:12      /tour/guide/query:2
00:00:13      /tour/guide/query:3
00:00:14      /tour/guide/query:6
00:00:15      /tour/guide/query:5
00:00:16      /tour/guide/query:5
00:00:17      /tour/guide/query:6
00:00:18      /tour/guide/query:3
00:00:19      /tour/guide/query:4
00:00:20      /tour/guide/query:4
00:00:21      /tour/guide/query:2
00:00:22      /tour/guide/query:6
00:00:23      /tour/guide/query:1
00:00:24      /tour/guide/query:3
00:00:25      /tour/guide/query:9
00:00:26      /tour/guide/query:6
00:00:27      /tour/guide/query:8
00:00:28      /tour/guide/query:3
00:00:29      /tour/guide/query:1
00:00:30      /tour/guide/query:1
00:00:31      /tour/guide/query:2
00:00:32      /tour/guide/query:10
00:00:33      /tour/guide/query:5
00:00:34      /tour/guide/query:6
00:00:35      /tour/guide/query:6
00:00:36      /tour/guide/query:3
00:00:37      /tour/guide/query:5
00:00:38      /tour/guide/query:7
00:00:39      /tour/guide/query:11
00:00:40      /tour/guide/query:22
00:00:41      /tour/guide/query:8
00:00:42      /tour/guide/query:5
00:00:43      /tour/guide/query:5
00:00:44      /tour/guide/query:6
```

(3) tour-produce-query

```

█ 终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
/tour/product/query:3154581
00:00:00      /tour/product/query:5
00:00:01      /tour/product/query:3
00:00:02      /tour/product/query:1
00:00:03      /tour/product/query:39
00:00:04      /tour/product/query:37
00:00:05      /tour/product/query:44
00:00:06      /tour/product/query:11
00:00:07      /tour/product/query:19
00:00:08      /tour/product/query:9
00:00:09      /tour/product/query:4
00:00:10      /tour/product/query:15
00:00:11      /tour/product/query:24
00:00:12      /tour/product/query:17
00:00:13      /tour/product/query:45
00:00:14      /tour/product/query:54
00:00:15      /tour/product/query:32
00:00:16      /tour/product/query:61
00:00:17      /tour/product/query:11
00:00:18      /tour/product/query:20
00:00:19      /tour/product/query:34
00:00:20      /tour/product/query:36
00:00:21      /tour/product/query:14
00:00:22      /tour/product/query:62
00:00:23      /tour/product/query:33
00:00:25      /tour/product/query:5
00:00:26      /tour/product/query:3
00:00:27      /tour/product/query:26
00:00:28      /tour/product/query:7
00:00:29      /tour/product/query:17
00:00:30      /tour/product/query:3
00:00:31      /tour/product/query:6
00:00:32      /tour/product/query:4
00:00:33      /tour/product/query:5
00:00:34      /tour/product/query:3
00:00:35      /tour/product/query:7
00:00:36      /tour/product/query:6
00:00:37      /tour/product/query:7
00:00:38      /tour/product/query:5
00:00:39      /tour/product/query:11
00:00:40      /tour/product/query:9
00:00:41      /tour/product/query:9
00:00:42      /tour/product/query:9
00:00:43      /tour/product/query:11
00:00:44      /tour/product/query:11
00:00:45      /tour/product/query:4

```

4. 统计每个接口(请求的URL)的平均响应时间，并且以接口为分组，按照小时时间窗，输出各个时间段各个接口平均的响应时间。每个接口的统计信息是一个文件，如接/tour/category/query的统计文件命名为：our-category-query.txt，每个文件的输出内容同上。

本部分结果在HDFS上的存储路径为：hdfs://master01:9000/user/2016st21/Stat/Out4。按照要求，每个接口的统计信息输出到相应的txt文件中（前面为总的平均响应时间，后面为分时间窗的平均响应时间），下面我们将展示几个txt文件的截图，其余的可在HDFS或者集群用户文件夹内查看：

(1) tour-category-ids-query

```
/tour/category/ids/query:7.829845603183698
0:00-1:00      /tour/category/ids/query:3.570709301298472
1:00-2:00      /tour/category/ids/query:3.0200811224150543
2:00-3:00      /tour/category/ids/query:3.9082923768691833
3:00-4:00      /tour/category/ids/query:4.81803649926735
4:00-5:00      /tour/category/ids/query:5.55997712978845
5:00-6:00      /tour/category/ids/query:4.487735634794459
6:00-7:00      /tour/category/ids/query:3.9245260379169666
7:00-8:00      /tour/category/ids/query:3.84271487334921
8:00-9:00      /tour/category/ids/query:4.065110176010135
9:00-10:00     /tour/category/ids/query:7.146962340900144
10:00-11:00    /tour/category/ids/query:14.134977233159233
11:00-12:00    /tour/category/ids/query:11.977461011407263
12:00-13:00    /tour/category/ids/query:5.0185422832150435
13:00-14:00    /tour/category/ids/query:8.041669984868998
14:00-15:00    /tour/category/ids/query:9.969534702365479
15:00-16:00    /tour/category/ids/query:9.228547248746526
16:00-17:00    /tour/category/ids/query:12.35620899827851
17:00-18:00    /tour/category/ids/query:7.532379174445136
18:00-19:00    /tour/category/ids/query:4.863724955172683
19:00-20:00    /tour/category/ids/query:4.7826674951569865
20:00-21:00    /tour/category/ids/query:4.476972692748206
21:00-22:00    /tour/category/ids/query:4.397135237676299
22:00-23:00    /tour/category/ids/query:4.462108594270487
23:00-24:00    /tour/category/ids/query:5.156549118387909
```

(2) tour-guide-query

```
□ 终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
/tour/guide/query:7.235481636137821
0:00-1:00      /tour/guide/query:3.7043178382464097
1:00-2:00      /tour/guide/query:3.452698441301018
2:00-3:00      /tour/guide/query:3.624633695505162
3:00-4:00      /tour/guide/query:3.7079622724281642
4:00-5:00      /tour/guide/query:3.9198552621448917
5:00-6:00      /tour/guide/query:3.6596845318860245
6:00-7:00      /tour/guide/query:3.764700581264363
7:00-8:00      /tour/guide/query:3.7179195679977264
8:00-9:00      /tour/guide/query:4.1177945984682225
9:00-10:00     /tour/guide/query:7.653479323761319
10:00-11:00    /tour/guide/query:17.39502789152936
11:00-12:00    /tour/guide/query:16.616495266634885
12:00-13:00    /tour/guide/query:5.153438334429782
13:00-14:00    /tour/guide/query:9.309448021413585
14:00-15:00    /tour/guide/query:12.058193154510619
15:00-16:00    /tour/guide/query:10.180944276348967
16:00-17:00    /tour/guide/query:14.842932412790697
17:00-18:00    /tour/guide/query:7.869683912771766
18:00-19:00    /tour/guide/query:5.205681129297653
19:00-20:00    /tour/guide/query:4.619422352448724
20:00-21:00    /tour/guide/query:4.367656255748078
21:00-22:00    /tour/guide/query:4.495573262505533
22:00-23:00    /tour/guide/query:4.691283113807903
23:00-24:00    /tour/guide/query:4.924494467760397
```

(3) tour-poi-query

```

□ 终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
/tour/poi/query:16.691687954411474
0:00-1:00      /tour/poi/query:6.368181818181818
1:00-2:00      /tour/poi/query:8.610169491525424
2:00-3:00      /tour/poi/query:6.962121212121212
3:00-4:00      /tour/poi/query:6.838709677419355
4:00-5:00      /tour/poi/query:9.050632911392405
5:00-6:00      /tour/poi/query:6.719298245614035
6:00-7:00      /tour/poi/query:5.262745098039216
7:00-8:00      /tour/poi/query:6.3254716981132075
8:00-9:00      /tour/poi/query:6.168103448275862
9:00-10:00     /tour/poi/query:28.71219512195122
10:00-11:00    /tour/poi/query:46.79024390243902
11:00-12:00    /tour/poi/query:31.259414225941423
12:00-13:00    /tour/poi/query:7.261904761904762
13:00-14:00    /tour/poi/query:29.55426356589147
14:00-15:00    /tour/poi/query:33.24271844660194
15:00-16:00    /tour/poi/query:22.776859504132233
16:00-17:00    /tour/poi/query:41.03557312252964
17:00-18:00    /tour/poi/query:22.61574074074074
18:00-19:00    /tour/poi/query:15.340425531914894
19:00-20:00    /tour/poi/query:11.986486486486486
20:00-21:00    /tour/poi/query:7.7976190476190474
21:00-22:00    /tour/poi/query:8.972222222222221
22:00-23:00    /tour/poi/query:9.061302681992338
23:00-24:00    /tour/poi/query:7.930232558139535

```

选取访问量较大的tour-guide-query接口，我们绘制出其24小时内响应时间的变化图了：



接口的响应时间与访问流量息息相关，也直接影响到了用户体验。故可以从响应时间趋势中挖掘出访问人数较多的时间，适当增加服务器资源和优化调控策略，提高网站的稳定性。

5. 设计预测算法来预测下一天（2015-09-22）每个小时窗内每个接口（请求的URL）的访问总频次。由前面一章可以看到，本任务可以分为(1)预测接口访问次数并输出预测结果和真实结果与(2)RMSE计算两个部分。两部分的结果分别为(以结果较好的线性回归为例做展示)：

(1) Step1 的部分输出结果（输出格式为：接口#小时真实值#预测值(若此部分只有一个数字则表示真实值为0)）的截图如下：(本部分结果在HDFS上的存储路径为：(hdfs://master01:9000/user/2016st21/Pred/avgout1 hdfs://master01:9000/user/2016st21/Pred/inter1out1 hdfs://master01:9000/user/2016st21/Pred/fitout1)

```

终端文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H) 14:50
/tour/category/query#00 79984#80842
/tour/category/query#01 120737#122368
/tour/category/query#02 317457#316682
/tour/category/query#03 558892#560743
/tour/category/query#04 562140#551047
/tour/category/query#05 333262#306907
/tour/category/query#06 96605#88110
/tour/category/query#07 118275#96066
/tour/category/query#08 150448#121186
/tour/category/query#09 170487#150483
/tour/category/query#10 191882#170925
/tour/category/query#11 193512#175511
/tour/category/query#12 174589#184511
/tour/category/query#13 199296#212657
/tour/category/query#14 206399#205750
/tour/category/query#15 196270#183490
/tour/category/query#16 204375#218093
/tour/category/query#17 172462#179323
/tour/category/query#18 158181#195610
/tour/category/query#19 176307#195607
/tour/category/query#20 209062#203610
/tour/category/query#21 240544#188294
/tour/category/query#22 235386#185690
/tour/category/query#23 183153#166398
/tour/hotelsuggestion/query#00 32#22
/tour/hotelsuggestion/query#01 27#41
/tour/hotelsuggestion/query#02 17#10
/tour/hotelsuggestion/query#03 4#7
/tour/hotelsuggestion/query#04 7#0
/tour/hotelsuggestion/query#05 9#1

终端文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H) 14:51
/tour/category/statis/query#00 381#366
/tour/category/statis/query#01 497#463
/tour/category/statis/query#02 425#459
/tour/category/statis/query#03 393#377
/tour/category/statis/query#04 1#1
/tour/category/statis/query#05 92#97
/tour/category/statis/query#06 1#0
/tour/category/statis/query#07 0#1
/tour/category/statis/query#08 1#2
/tour/category/statis/query#09 47#1
/tour/category/statis/query#10 22#2
/tour/category/statis/query#11 32#10
/tour/category/statis/query#12 23#6
/tour/category/statis/query#13 24#10
/tour/category/statis/query#14 79#8
/tour/category/statis/query#15 108#4
/tour/category/statis/query#16 217#2
/tour/category/statis/query#17 262#5
/tour/category/statis/query#18 24#2
/tour/category/statis/query#19 10#1
/tour/category/statis/query#20 4#3
/tour/category/statis/query#21 11#0
/tour/category/statis/query#22 112#105
/tour/category/statis/query#23 3#0
/tour/em-task/exec#01 2#2
/tour/guide/delete#09 0#0
/tour/guide/delete#10 5#3
/tour/guide/delete#11 0#1
/tour/guide/delete#12 0#0
/tour/guide/delete#13 0#0

```

(2) Step2: 从Step1的输出结果中计算RMSE的结果如下：(本部分结果在HDFS上的存储路径为： hdfs://master01:9000/user/2016st21/Pred/avgout2 hdfs://master01:9000/user/2016st21/Pred/inter1out2 hdfs://master01:9000/user/2016st21/Pred/fitout2)

平均值预测的RMSE结果为：

```
终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
RMSE      11776.069976733326
~
```

插值预测的RMSE结果为：

```
终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
RMSE      3637.6285158697756
~
```

线性回归的RMSE结果为：

```
终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
RMSE      3500.015422959075
~
```

对于预测结果的分析与讨论，请查看下一章节。

6. 扩展预测分析任务：在本部分，我们统计了2015-09-08 一天内各个IP对于各个接口的访问总次数，从而可以进行进一步的分析来产生作用（对于其他时间的分析可通过程序输入参数而改变）。

本任务的输出结果的部分截图如下：(本部分结果在HDFS上的存储路径为：

hdfs://master01:9000/user/2016st21/Ana/Ana1)

输出格式为 (IP: 接口访问次数)

```
10.10.0.120:/tour/category/ids/query      21
10.10.0.120:/tour/category/query       35
10.10.0.120:/tour/guide/query        7
10.10.0.120:/tour/hotel-search/nearby-scenic/query      2
10.10.0.120:/tour/hotel-search/query     6
10.10.0.120:/tour/hotelsuggestion/query 4
10.10.0.120:/tour/phoenix/product/query 5
10.10.0.120:/tour/poi/query/queryProvinceList   1
10.10.0.120:/tour/poi/query/queryScenicNumPerCity    1
10.10.0.120:/tour/poi/query/queryScenicSpotCount    1
10.10.0.120:/tour/product/query 40
10.10.0.92:/tour/category/ids/query      2
10.10.0.92:/tour/category/query 257
10.10.0.92:/tour/poi/query      15
10.10.0.92:/tour/suggestion/query     20
172.22.49.35:/tour/hotel-search/query 3029
172.22.49.35:/tour/phoenix/product/query 814
172.22.49.46:/tour/category/ids/query 124645
172.22.49.46:/tour/category/query 462757
172.22.49.46:/tour/category/statis/query 21
172.22.49.46:/tour/category/tuniu-hot/query 5
172.22.49.46:/tour/category/vendor-statis/query 2132
172.22.49.46:/tour/guide/query 108843
```

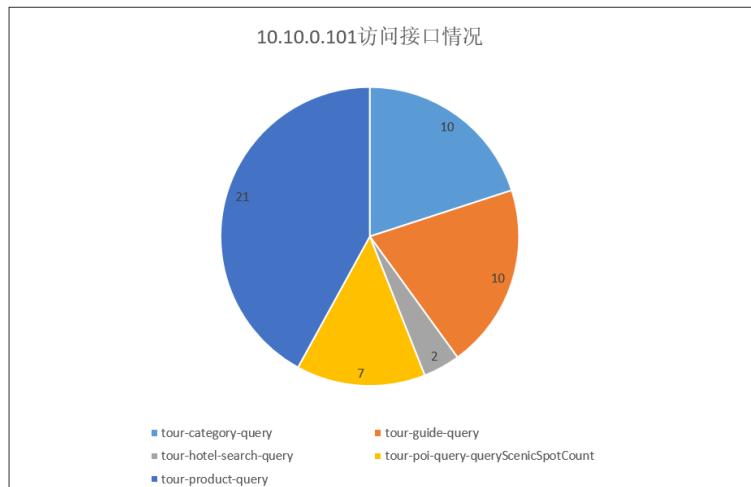
```

X - □ 终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
172.22.49.45:/tour/category/ids/query 123036
172.22.49.45:/tour/category/query 458592
172.22.49.45:/tour/category/statis/query 24
172.22.49.45:/tour/category/tuniu-hot/query 12
172.22.49.45:/tour/category/vendor-statis/query 2027
172.22.49.45:/tour/guide/query 108167
172.22.49.45:/tour/guide/update 31
172.22.49.45:/tour/hotel-search/nearby-scenic/query 2617
172.22.49.45:/tour/hotel-search/query 25056
172.22.49.45:/tour/hotelSuggestion/query 403
172.22.49.45:/tour/phoenix/product/query 14924
172.22.49.45:/tour/poi/query 357
172.22.49.45:/tour/poi/query/queryCategory 1396
172.22.49.45:/tour/poi/query/queryProvinceList 2
172.22.49.45:/tour/poi/query/queryScenicNumPerCity 233
172.22.49.45:/tour/poi/query/queryScenicSpotCount 48229
172.22.49.45:/tour/poi/query/queryScenicTypeList 1
172.22.49.45:/tour/poi/scenictype/provincelist/query 1
172.22.49.45:/tour/product/query 544849
172.22.49.45:/tour/suggestion/query 14356
172.22.49.56:/tour/category/ids/query 4353
172.22.49.56:/tour/category/query 125733
172.22.49.56:/tour/category/weekendproduct/query 3920

```

通过上面的输出结果，我们可以进行以下的一些分析：

我们随机选取了10.10.0.101这个IP绘制出其在2015-09-08 一天内访问不同接口的饼状图如下：



从图中我们可以看出，该IP的用户对于哪些接口的访问占比较多，这也可以从侧面折射出用户的某种偏好。如果说第二个任务是通过对日志中IP的访问次数行为分析得出用户的活跃时间和时间维度方面的数据，那本任务则是通过分析IP获得用户兴趣和消费倾向。两者结合起来，用户的画像就更加清晰了。因此对日志中的IP记录进行追踪，对于优化用户体验、增加用户停留时间和刺激消费行为等方面都有重大的意义，这也正是大数据的精华和魅力所在。

当然，除了针对用户的数据挖掘方面的应用。分析日志中IP对接口的访问情况也可以应用于网站安全和稳定性维护中。比如若某个IP平时对一个接口访问极少，但在某个时间内突然出现反常的爆发的大量访问行为，这可能导致网站运行异常甚至崩溃，那这时极有可能发生了针对服务器的DDOS攻击。通过日志分析，可以将对某些接口的一些异常的访问IP进行

过滤或者限定其单位时间内的访问请求次数，这也是一个防DDOS攻击的安全软件可以采用的策略。

§5. 实验优化与性能分析

5.1 实验优化与优化效果展示

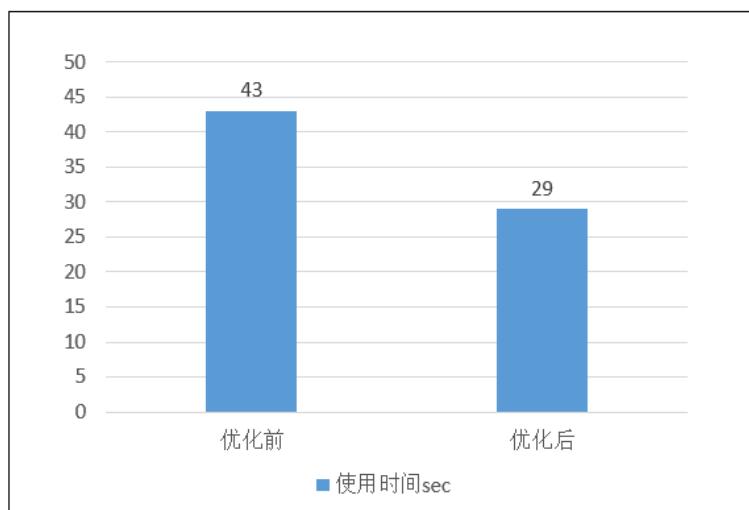
在本次实验的中我们尝试的优化可以主要分为以下三种手段：(1) 通用的优化手段；(2) 统计型任务的优化；(3) 预测型任务的优化。

5.1.1 通用优化手段

在本次实验中，我们采用的通用优化方式有：

1. 在实验设计过程中，所有任务都尽量能够在1个JOB中即可完成任务，避免了执行多个JOB。具体体现在预测型任务中将预测值与真实值组合为同一个键值输出。从而在计算RMSE时操作更为简单，也可以很大程度上减少计算时间。
2. 在每个JOB中尽可能的实现了combiner类从而提高计算效率与减少传输带宽。

尤其在第一个任务中，设置了combiner类可以在很大程度上减少网络数据传输量，提高了系统效率。使得程序运行时间有了较为明显的减少，见下图：



此外，在后面的线性回归过程中，使用Combiner也避免了在Reduce阶段的大量重复的计算，包括对字符串与数组的操作。

5.1.2 统计型任务的优化

统计型任务均为类似于 wordcount 的方法进行统计，因而其算法优化空间很小。因而本部分，我们的优化主要在以下两个部分：

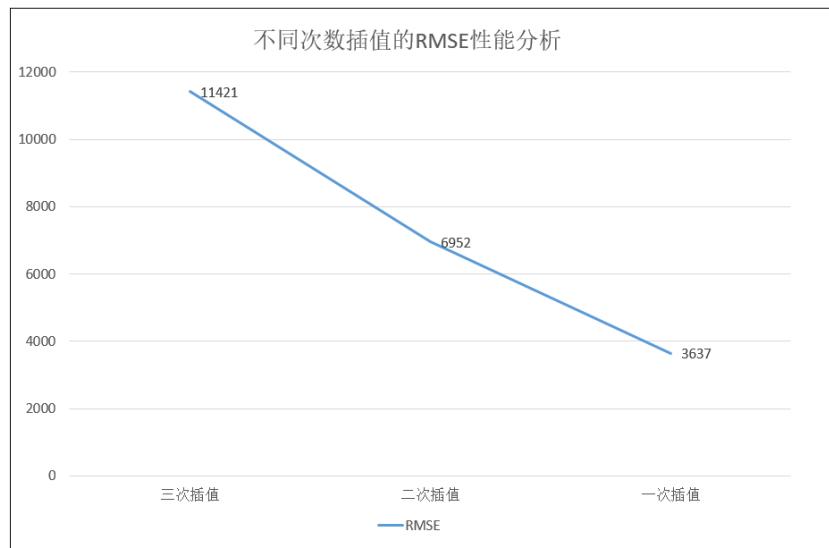
1. **设置 Reduce 任务的数目：**由于第一个任务只有一个输出文件，因而本方法无效。对于其他几个任务，通过设置较多的 Reduce 数目可以让每个信息单元的统计尽可能在不同节点上运行，提高并行化程度，从而缩短运行时间。
2. **日志记录的切割过程：**通过查看日志的具体格式，我们可以看到日志记录的以空格进行切割所得字符串序列的长度并不一定。由于可能出现部分信息的缺失，记录分段可能会出现变化，因而首先要判断记录是哪种类型才能进行正确的切割从而统计。这会涉及到很多的字符串与数字的比较，带来大量的操作。通过进一步的分析，我们发现可以通过正向及逆向两个方向定位来快速切割出所需信息，而不必通过较多操作来判断记录是哪种类型。例如：访问IP总是记录按照空格切割所得字符串数组的0号元素，而请求响应时间总是数组的最后一个元素，状态码总是该数组的倒数第三个元素。
因而，借助于数组长度这个值，我们可以将不同类型的记录的切割统一起来。这样便可避免判断记录类型所可能带来的大量字符串的操作。

5.1.3 预测型任务的优化

由于对预测型任务存在评价标准——RMSE，因而预测型任务优化的主要方向便是算法上的优化，效率上的优化在这部分较为次要。

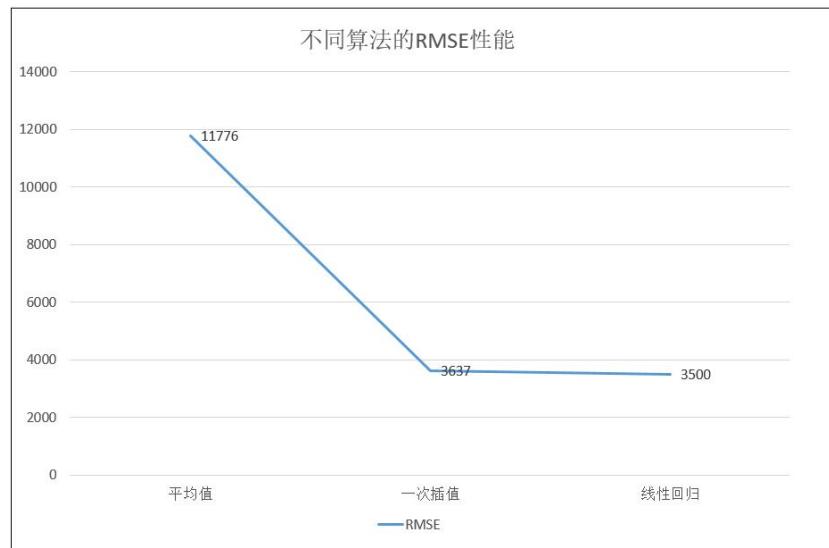
面对这样的任务，取平均值可以说是最简单的方法，也是我们最容易想到的方法。我们首先使用这个方法作为baseline。后面的方法将用来和次方法进行比较。在前面一章，我们可以看到使用平均值法进行实验，预测结果与真实值做RMSE验证得到的结果为11776。

在此基础上，陈越琦同学提出了使用多项式插值来进行预测的方法。因而，我们实现了使用拉格朗日多项式插值来进行插值的算法。为确定插值应该使用的多项式次数，我们进行了一些尝试，分别得到对应预测结果的RMSE值，如下图：



由上图可见，由于数据本身的特性，最简单的模型反而获得了最好的效果的。因而，我们使用了一次多项式插值来作为最终的预测方法。

在前面的基础上我们又尝试了线性回归进行预测的方法，最终获得了比一次多项式插值稍好的结果，三种方法所得RMSE值的比较见下图：



由上可见，最终我们获得了比平均值方法好很多的RMSE验证效果。采用一次多项式插值与线性回归预测均可以获得可以接受的较好的预测效果。而且使用插值的方法可以减少所需的数据量，从而大幅度减少中间数据的传输时间，提高预测速度。

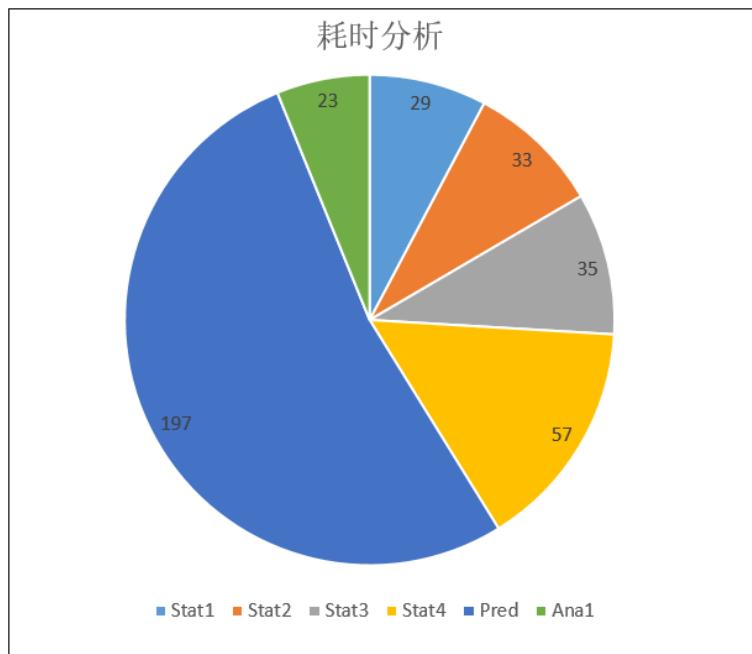
5.2 实验性能分析

1. 运行时间分析

根据集群的运行记录，我们可以得到几个任务的运行时间如下：

task	Time(sec)	Percentage(%)
Stat1	29	7.75
Stat2	33	8.82
Stat3	35	9.36
Stat4	57	15.24
Pred	197	52.68
Ana1	23	6.15

将数据按照饼状图进行展示，可以得到如下结果：



由上面的图可以看到，Stat1-Stat4 和 Ana1 所需时间较短，Pred 过程需要运行较长的时间。这也比较好理解，Stat1-Stat4 和 Ana1 任务只处理2015-09-08.log 的数据，时间相对较短，其中在 Stat4 任务中可能由于集群情况，比其他程序略慢。而 Pred 处理了所有15天的日志文件数据，因而运行时间明显更长，尤其是在 Step1。程序运行速度虽然受到集群任务多少和计算资源的波动影响，但整体的速度都是比较快的，多次测试下来运行速度较为稳定。可以看到由于我们前面进行了算法与代码的优化，程序的整体运行速度还是比较快的。

2. 程序可扩展性分析

在整体设计过程中，我们保持了比较好的模块化风格，程序的可扩展性较强。很多模块能被其他程序直接使用或者进行简单修改即可使用，例如RMSE的计算程序可被重复利用，以及

统计型任务整体只可在在一个任务的程序上进行较少修改便可实现。

3. 实验的进一步扩展方向

日志分析有很多应用，可以从中发掘很多信息，我们可以在前面的基础上设计更多的程序来进行进一步地挖掘有用的信息。同时，通过对数据的更深层次的规律探索，通过更好的模型，可能会获得更好更准确的预测结果。

§6. 实验总结

6.1 实验内容总结

本次实验是大数据处理综合实验的课程设计实验。经过仔细权衡实现难度和时间，考虑到我们组员各自的投入时间和需求。我们最终在三个题目以及自拟题目中做出了完成“日志统计分析”的选择。在本次综合实验中，我们将主要实现日志数据的统计分析和基于Mapreduce的预测模型设计，通过对历史日志数据的分析建立预测模型，并在完成基本任务的要求上做一些拓展和创新。我们完成了以下几个任务：

- 统计日志中各个状态码出现的频次，并按照小时时间窗、输出各个时间段状态码的统计情况。
- 统计每个IP访问总的频次，并且按照小时时间窗，输出各个时间段各个IP访问的情况。
- 统计每个接口（请求的URL）访问总的频次，并且以接口名为文件名，按照秒为单位的时间窗，统计各个时间段各接口的访问情况。
- 统计每个接口的平均响应时间，并且以接口为分组，按照小时时间窗、输出各个时间段各个接口平均的响应时间。
- 接口访问频次预测，给2015-09-08.log到2015-09-21.log共14天的日志文件，作为训练数据，设计了不同的预测算法（平均值、一次插值、两次插值、三次插值、拟合）来预测2015-09-22.log一个小时时间窗每个接口的访问总频次，并计算不同算法获得的RMSE，对比方法优劣并进行改进优化。

6.2 团队合作总结

在本次课程设计实验中，通过前面几次平时实验的磨合，合作过程较为顺利。遵循“由易到难、由简到繁、多路并行”的原则进行分工。在算法和模型设计阶段。依靠平时实验积累的经验，快速确立了统计任务部分（前四个要求）的解题思路，同步开始编程工作；与此同时对较难的预测任务部分，让每个人通过阅读一些文章与博客，找助教探讨，搜索引擎等方式

进行准备，最终获得了不同的方法，从最简单的平均值法，到较为复杂的插值法和回归算法，都进行了尝试，并进行了充分的验证。在代码编写过程中，充分利用Github进行团队协作与管理，保证代码的实时共享和快速交流。在实验报告编写中，每个人对自己所属的任务的设计原理、伪代码等进行了阐述，并安排专人对实验结果、执行报告、分析图表等成果性内容进行整理和收集，最终合并实验报告进行美化排版，形成了最终的成果。

6.3 实验任务分工

本次实验的分工如下：

陈越琦	完成预测型任务的平均值法与插值法设计与代码编写； 提供拓展分析部分思路； 完成对应部分实验报告初稿； 完善实验报告初稿.
刘威	完成统计型任务的设计与代码编写； 实现预测分析的线性回归法； 完成对应部分以及实验优化部分实验报告的初稿； 设计实验报告结构； 汇总每个人编写的实验报告得到初稿.
杨杰才	协助刘威和陈越琦进行代码的测试与错误查找； 对实验运行的结果与数据进行汇总与截图； 完成实验报告的性能分析和实验结果部分的初稿； 完成实验报告的图表绘制.
周子博	协助杨杰才完成实验数据的统计，汇总与截图； 协助刘威进行实验报告初稿的排版和统计型任务的测试； 协助陈越琦完成预测型任务的Driver编写
共同完成	实验思路探讨； 实验优化方法探讨； 实验报告校对与修改.

组员分工表

§7. 附录：各任务执行报告截图

7.1 统计型任务

统计型任务共有四个，四个任务的执行报告的截图分别为：

1. 任务 1(Stat 1)

(1) 集群All Application (<http://114.212.190.91:8088/>) WebUI页面中查看Job的执行状态

Scheduler Metrics		Scheduling Resource Type			Minimum Allocation			Maximum Allocation		
Fair Scheduler	[MEMORY, CPU]				<memory:1024, vCores:1>			<memory:8192, vCores:8>		
Show 100 ▾ entries					Search:					
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1467969296998_6142	2016st21	Stat4	MAPREDUCE	root.default	Mon Jul 18 14:43:09 +0800 2016	Mon Jul 18 14:44:06 +0800 2016	FINISHED	SUCCEEDED		History
application_1467969296998_6141	2016st21	Stat3	MAPREDUCE	root.default	Mon Jul 18 14:42:32 +0800 2016	Mon Jul 18 14:43:07 +0800 2016	FINISHED	SUCCEEDED		History
application_1467969296998_6138	2016st21	Stat2	MAPREDUCE	root.default	Mon Jul 18 14:41:57 +0800 2016	Mon Jul 18 14:42:30 +0800 2016	FINISHED	SUCCEEDED		History
application_1467969296998_6136	2016st21	Stat1	MAPREDUCE	root.default	Mon Jul 18 14:41:25 +0800 2016	Mon Jul 18 14:41:54 +0800 2016	FINISHED	SUCCEEDED		History

(2) WebUIx页面 (<http://114.212.190.91:19888/jobhistory>)

Show 100 ✓ entries										Search:			
Submit	Start Time	Finish	Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total	Reduces Completed	
2016-01-28 14:41:25	2016-01-28 14:41:34	2016-01-28 14:41:34	job_1467969296998_6136	Stat1	2016st21	root.default	SUCCEEDED	8	8	1	1		

根据Job ID链接进入Job详细页面，如下所示：

Counters for job_1467969296998_6136						
Counter Group	Name	Counters	Map	C	Reduce	Total
File System Counters	FILE_Number of bytes read	0	1,869	0	1,869	1,869
	FILE_Number of bytes written	922,407	117,526	0	1,046,033	1,046,033
	FILE_Number of large read operations	0	0	0	0	0
	FILE_Number of read operations	0	0	0	0	0
	FILE_Number of write operations	0	0	0	0	0
	HDFS_Number of bytes read	970,826,937	0	0	0	970,826,937
	HDFS_Number of bytes written	0	908	0	908	908
	HDFS_Number of large read operations	0	0	0	0	0
	HDFS_Number of read operations	24	3	0	27	27
	HDFS_Number of write operations	0	2	0	2	2
Job Counters	Data-local map tasks	0	0	0	0	4
	Killed map tasks	0	0	0	0	1
	Launched map tasks	0	0	0	0	9
	Launched reduce tasks	0	0	0	0	1
	Local-locally map tasks	0	0	0	0	5
	Total negative seconds taken by all map tasks	0	0	0	0	95,649,472
	Total negative-seconds taken by all reduce tasks	0	0	0	0	7,116,800
	Total time spent by all map tasks (ms)	0	0	0	0	93,603
	Total time spent by all maps in occupied slots (ms)	0	0	0	0	93,603
	Total time spent by all maps in unoccupied slots (ms)	0	0	0	0	0,959
	Total time spent by all reduces in occupied slots (ms)	0	0	0	0	6,950
Map-Reduce Framework	Total vcore-seconds taken by all map tasks	0	0	0	0	93,603
	Total vcore-seconds taken by all reduce tasks	0	0	0	0	6,950
	Combine input records	20,819,052	0	0	0	20,819,052
	Combine output records	129	0	0	0	129
	CPU Time spent (ms)	79,150	2,040	0	0	81,190
	Failed Shuffles	0	0	0	0	0
	GC time elapsed (ms)	2,724	75	0	0	2,799
	Input split bytes	1,000	0	0	0	1,000
	Map input records	10,409,526	0	0	0	10,409,526
	Map output bytes	239,419,098	0	0	0	239,419,098
Shuffle Errors	Map output normalized bytes	1,911	0	0	0	1,911
	Map output records	20,819,052	0	0	0	20,819,052
	Merged Map outputs	0	0	0	0	0
	Physical memory (bytes) snapshot	2,138,562,560	168,329,216	0	0	2,306,891,776
	Reduce input groups	0	75	0	0	75
	Reduce input records	0	129	0	0	129
	Reduce input records	0	27	0	0	27
	Reduce shuffle bytes	0	1,911	0	0	1,911
	Shuffled Maps	0	0	0	0	0
	Spilled Records	129	129	0	0	258
File Input Format Counters	Total committed heap usage (bytes)	1,658,322,944	201,326,592	0	0	1,859,649,536
	Virtual memory (bytes) snapshot	13,099,524,096	1,652,158,464	0	0	14,751,682,560
File Output Format Counters	Name	Counters	Map	C	Reduce	Total
	Bytes Read	970,825,937	0	0	0	970,825,937
	Name	Counters	Map	C	Reduce	Total
	Bytes Written	0	908	0	0	908
	Name	Counters	Map	C	Reduce	Total
	Bytes Written	0	908	0	0	908

2. 任务 2(Stat 2)

(1) 集群All Application (<http://114.212.190.91:8088/>) WebUI页面中查看Job的执行状态

Scheduler Metrics										
Scheduler Type		Scheduling Resource Type		Minimum Allocation			Maximum Allocation			
Fair Scheduler	[MEMORY, CPU]	<memory:1024, vCores:1>			<memory:8192, vCores:8>				Search:	
Show 100 ▾ entries	entries									
application_1467969296998_6142	2016st21 Stat4	MAPREDUCE	root.default	Mon Jul 18 18 14:43:09 +0800 2016	Mon Jul 18 18 14:44:06 +0800 2016	FINISHED	SUCCEEDED			History
application_1467969296998_6141	2016st21 Stat3	MAPREDUCE	root.default	Mon Jul 18 18 14:42:32 +0800 2016	Mon Jul 18 18 14:43:07 +0800 2016	FINISHED	SUCCEEDED			History
application_1467969296998_6138	2016st21 Stat2	MAPREDUCE	root.default	Mon Jul 18 18 14:41:57 +0800 2016	Mon Jul 18 18 14:42:30 +0800 2016	FINISHED	SUCCEEDED			History
application_1467969296998_6136	2016st21 Stat1	MAPREDUCE	root.default	Mon Jul 18 18 14:41:25 +0800 2016	Mon Jul 18 18 14:41:54 +0800 2016	FINISHED	SUCCEEDED			History

(2) WebUIx页面 (<http://114.212.190.91:19888/jobhistory>)

Search:										
Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total
2016-07-18 14:41:57 CST	2016-07-18 14:42:03 CST	2016-07-18 14:42:30 CST	job_1467969296998_6138	Stat2	2016st21	root.default	SUCCEEDED	8	8	8

根据Job ID链接进入Job详细页面，如下所示：

loop Counters for job_1467969296998_6138										
Counter Group		Counters								
		Name	Map	Reduce	Total					
File System Counters	FILE: Number of bytes read	243,640	53,936	297,576						
	FILE: Number of bytes written	1,040,382	985,744	2,026,126						
	FILE: Number of large read operations	0	0	0						
	FILE: Number of read operations	0	0	0						
	FILE: Number of write operations	0	0	0						
	HDFS: Number of bytes read	970,826,937	0	970,826,937						
	HDFS: Number of bytes written	0	29,204	29,204						
	HDFS: Number of large read operations	0	0	0						
	HDFS: Number of read operations	24	24	48						
	HDFS: Number of write operations	0	78	78						
Job Counters	Data-local map tasks	0	0	6						
	Killed map tasks	0	0	1						
	Launched map tasks	0	0	9						
	Launched reduce tasks	0	0	8						
	Rack-local map tasks	0	0	3						
	Total megabyte-seconds taken by all map tasks	0	0	127,991,808						
	Total megabyte-seconds taken by all reduce tasks	0	0	46,172,160						
	Total time spent by all map tasks (ms)	0	0	124,992						
	Total time spent by all maps in occupied slots (ms)	0	0	124,992						
	Total time spent by all reduce tasks (ms)	0	0	45,090						
Map-Reduce Framework	Total time spent by all reduces in occupied slots (ms)	0	0	45,090						
	Total vcore-seconds taken by all map tasks	0	0	124,992						
	Total vcore-seconds taken by all reduce tasks	0	0	45,090						
	Combine input records	20,819,052	0	20,819,052						
	Combine output records	2,358	0	2,358						
	CPU time spent (ms)	118,320	18,320	136,640						
	Failed Shuffles	0	0	0						
	GC time elapsed (ms)	4,130	529	4,659						
	Input split bytes	1,000	0	1,000						
	Map input records	10,409,526	0	10,409,526						
Shuffle Errors	Map input bytess	426,741,372	0	426,741,372						
	Map output materialized bytes	54,272	0	54,272						
	Map output records	20,819,052	0	20,819,052						
	Merged Map outputs	0	64	64						
	Physical memory (bytes) snapshot	2,164,551,680	1,418,547,200	3,583,098,880						
	Reduce input groups	0	1,029	1,029						
	Reduce input records	0	2,358	2,358						
File Input Format Counters	Reduce output records	0	0	0						
	Reduce shuffle bytes	0	54,272	54,272						
	Shuffled Maps	0	64	64						
	Spilled Records	4,626	2,358	6,984						
File Output Format Counters	Total committed heap usage (bytes)	1,668,284,416	1,610,612,736	3,278,897,152						
	Virtual memory (bytes) snapshot	13,166,665,728	13,175,799,808	26,342,465,536						
	Bytes Written	0	0	0						

3. 任务 3(Stat 3)

(1) 集群All Application (<http://114.212.190.91:8088/>) WebUI页面中查看Job的执行状态

Scheduler Metrics										
Scheduler Type	Scheduling Resource Type		Minimum Allocation			Maximum Allocation				
Fair Scheduler	[MEMORY, CPU]		<memory:1024, vCores:1>			<memory:8192, vCores:8>				
Show 100 ▾ entries	Search:									
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1467969296998_6142	2016st21	Stat4	MAPREDUCE	root.default	Mon Jul 18 14:43:09 +0800 2016	Mon Jul 18 14:44:06 +0800 2016	FINISHED	SUCCEEDED		History
application_1467969296998_6141	2016st21	Stat3	MAPREDUCE	root.default	Mon Jul 18 14:42:32 +0800 2016	Mon Jul 18 14:43:07 +0800 2016	FINISHED	SUCCEEDED		History
application_1467969296998_6138	2016st21	Stat2	MAPREDUCE	root.default	Mon Jul 18 14:41:57 +0800 2016	Mon Jul 18 14:42:30 +0800 2016	FINISHED	SUCCEEDED		History
application_1467969296998_6136	2016st21	Stat1	MAPREDUCE	root.default	Mon Jul 18 14:41:25 +0800 2016	Mon Jul 18 14:41:54 +0800 2016	FINISHED	SUCCEEDED		History

(2) WebUIx页面 (<http://114.212.190.91:19888/jobhistory>)

Search:											
Show 100 ▾ entries	Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total
2016-07-18 14:42:32 CST	2016-07-18 14:42:38 CST	2016-07-18 14:43:07 CST	job_1467969296998_6141	Stat3	2016st21	root.default	SUCCEEDED	8	8	8	8

根据Job ID链接进入Job详细页面，如下所示：

loop Counters for job_1467969296998_6141

Counter Group	Name	Counters			Total
		Map	Reduce	Total	
File System Counters	FILE: Number of bytes read	27,953,156	29,100,145	57,053,301	
	FILE: Number of bytes written	57,819,603	30,031,985	87,851,588	
	FILE: Number of large read operations	0	0	0	
	FILE: Number of read operations	0	0	0	
	FILE: Number of write operations	0	0	0	
	HDFS: Number of bytes read	970,826,937	0	970,826,937	
	HDFS: Number of bytes written	0	25,877,981	25,877,981	
	HDFS: Number of large read operations	0	0	0	
	HDFS: Number of read operations	24	24	48	
	HDFS: Number of write operations	0	45	45	
Job Counters	Data-local map tasks	0	0	6	
	Killed map tasks	0	0	1	
	Launched map tasks	0	0	9	
	Launched reduce tasks	0	0	8	
	Rack-local map tasks	0	0	3	
	Total megabyte-seconds taken by all map tasks	0	0	135,528,448	
	Total megabyte-seconds taken by all reduce tasks	0	0	102,598,656	
	Total time spent by all map tasks (ms)	0	0	132,352	
	Total time spent by all maps in occupied slots (ms)	0	0	132,352	
	Total time spent by all reduce tasks (ms)	0	0	100,194	
Map-Reduce Framework	Total time spent by all reduces in occupied slots (ms)	0	0	100,194	
	Total vcore-seconds taken by all map tasks	0	0	132,352	
	Total vcore-seconds taken by all reduce tasks	0	0	100,194	
	Combine input records	20,819,052	0	20,819,052	
	Combine output records	686,609	0	686,609	
	CPU time spent (ms)	138,790	53,990	192,780	
	Failed Shuffles	0	0	0	
	GC time elapsed (ms)	3,285	1,275	4,560	
	Input split bytes	1,000	0	1,000	
	Map input records	10,409,526	0	10,409,526	
Shuffle Errors	Map output bytes	668,793,926	0	668,793,926	
	Map output materialized bytes	29,100,481	0	29,100,481	
	Map output records	20,819,052	0	20,819,052	
	Merged Map outputs	0	64	64	
	Physical memory (bytes) snapshot	2,142,887,936	1,495,732,224	3,638,620,160	
	Reduce input groups	0	685,952	685,952	
	Reduce input records	0	686,609	686,609	
	Reduce output records	0	0	0	
	Reduce shuffle bytes	0	29,100,481	29,100,481	
	Shuffled Maps	0	64	64	
File Input Format Counters	Spilled Records	1,342,086	686,609	2,028,695	
	Total committed heap usage (bytes)	1,669,332,992	1,591,738,368	3,261,071,360	
	Virtual memory (bytes) snapshot	13,166,526,464	13,164,023,808	26,330,550,272	
	BAD_ID	0	0	0	
File Output Format Counters	CONNECTION	0	0	0	
	IO_ERROR	0	0	0	
	WRONG_LENGTH	0	0	0	
	WRONG_MAP	0	0	0	
Bytes Read	WRONG_REDUCE	0	0	0	
	Name	970,825,937	0	970,825,937	
Bytes Written	Name	0	0	0	
	Bytes Written	0	0	0	

4. 任务 4(Stat 4)

(1) 集群All Application (<http://114.212.190.91:8088/>) WebUI页面中查看Job的执行状态

Scheduler Metrics										
Scheduler Type	Scheduling Resource Type		Minimum Allocation			Maximum Allocation				
Fair Scheduler	[MEMORY, CPU]		<memory:1024, vCores:1>			<memory:8192, vCores:3>				
Show 100 ▾ entries										
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1467969296998_6142	2016st21	Stat4	MAPREDUCE	root.default	Mon Jul 18 14:43:09 +0800 2016	Mon Jul 18 14:44:06 +0800 2016	FINISHED	SUCCEEDED		History
application_1467969296998_6141	2016st21	Stat3	MAPREDUCE	root.default	Mon Jul 18 14:42:32 +0800 2016	Mon Jul 18 14:43:07 +0800 2016	FINISHED	SUCCEEDED		History
application_1467969296998_6138	2016st21	Stat2	MAPREDUCE	root.default	Mon Jul 18 14:41:57 +0800 2016	Mon Jul 18 14:42:30 +0800 2016	FINISHED	SUCCEEDED		History
application_1467969296998_6136	2016st21	Stat1	MAPREDUCE	root.default	Mon Jul 18 14:41:25 +0800 2016	Mon Jul 18 14:41:54 +0800 2016	FINISHED	SUCCEEDED		History

(2) WebUIx页面 (<http://114.212.190.91:19888/jobhistory>)

Search:											
Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total	Reduces Completed
2016.07.18 14:43:09 CST	2016.07.18 14:43:15 CST	2016.07.18 14:44:06 CST	job_1467969296998_6142	Stat4	2016st21	root.default	SUCCEEDED	8	8	8	8

根据Job ID链接进入Job详细页面，如下所示：

loop Counters for job_1467969296998_6142					
Counter Group		Counters			
	Name	Map	Reduce	Total	
File System Counters	FILE: Number of bytes read	627,329,672	647,974,958	1,275,304,630	
	FILE: Number of bytes written	1,276,030,679	648,905,638	1,924,936,317	
	FILE: Number of large read operations	0	0	0	
	FILE: Number of read operations	0	0	0	
	FILE: Number of write operations	0	0	0	
	HDFS: Number of bytes read	970,826,937	0	970,826,937	
	HDFS: Number of bytes written	0	32,181	32,181	
	HDFS: Number of large read operations	0	0	0	
	HDFS: Number of read operations	24	24	48	
	HDFS: Number of write operations	0	45	45	
Job Counters	Data-local map tasks	0	0	8	
	Killed map tasks	0	0	1	
	Killed reduce tasks	0	0	1	
	Launched map tasks	0	0	9	
	Launched reduce tasks	0	0	9	
	Rack-local map tasks	0	0	1	
	Total megabyte-seconds taken by all map tasks	0	0	192,699,392	
	Total megabyte-seconds taken by all reduce tasks	0	0	212,755,456	
	Total time spent by all map tasks (ms)	0	0	188,183	
	Total time spent by all maps in occupied slots (ms)	0	0	188,183	
	Total time spent by all reduce tasks (ms)	0	0	207,769	
	Total time spent by all reduces in occupied slots (ms)	0	0	207,769	
	Total vcore-seconds taken by all map tasks	0	0	188,183	
Map-Reduce Framework	Total vcore-seconds taken by all reduce tasks	0	0	207,769	
	Combine input records	0	0	0	
	Combine output records	0	0	0	
	CPU time spent (ms)	169,890	79,480	249,370	
	Failed Shuffles	0	0	0	
	GC time elapsed (ms)	3,969	1,457	5,426	
	Input split bytes	1,000	0	1,000	
	Map input records	10,409,526	0	10,409,526	
	Map output bytes	606,336,770	0	606,336,770	
	Map output materialized bytes	647,975,258	0	647,975,258	
	Map output records	20,819,052	0	20,819,052	
	Merged Map outputs	0	64	64	
	Physical memory (bytes) snapshot	2,135,031,808	1,810,190,336	3,945,222,144	
	Reduce input groups	0	607	607	
	Reduce input records	0	20,819,052	20,819,052	
	Reduce output records	0	0	0	
Shuffle Errors	Reduce shuffle bytes	0	647,975,258	647,975,258	
	Shuffled Maps	0	64	64	
	Spilled Records	40,968,418	20,819,052	61,787,470	
	Total committed heap usage (bytes)	1,668,808,704	1,534,590,976	3,203,399,680	
	Virtual memory (bytes) snapshot	13,099,708,416	13,019,578,368	26,119,286,784	
	BAD_ID	0	0	0	
	CONNECTION	0	0	0	
File Input Format Counters	IO_ERROR	0	0	0	
	WRONG_LENGTH	0	0	0	
	WRONG_MAP	0	0	0	
File Output Format Counters	WRONG_REDUCE	0	0	0	
	Bytes Read	970,825,937	0	970,825,937	
	Bytes Written	0	0	0	

7.2 预测型任务

预测型任务分为两个步骤 Step1 与 Step2，分别为访问频次预测与RMSE值的计算。下面为两个步骤的执行报告：

1. Step 1

(1) 集群All Application (<http://114.212.190.91:8088/>) WebUI页面中查看Job的执行状态

Scheduler Metrics										
Scheduler Type	Scheduling Resource Type		Minimum Allocation			Maximum Allocation				
Fair Scheduler	[MEMORY, CPU]		<memory:1024, vCores:1>			<memory:8192, vCores:8>				
Show 100 ▾ entries									Search:	
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1467969296998_6859	2016st21	Prediction Step2	MAPREDUCE	root.default	Tue Jul 19 15:43:22 +0800 2016	Tue Jul 19 15:43:47 +0800 2016	FINISHED	SUCCEEDED		History
application_1467969296998_6856	2016st21	Prediction Step1	MAPREDUCE	root.default	Tue Jul 19 15:42:30 +0800 2016	Tue Jul 19 15:42:43 +0800 2016	FINISHED	SUCCEEDED		History

(2) WebUIx页面 (<http://114.212.190.91:19888/jobhistory>)

Search:										
Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total
2016.07.19 15:40:30 CST	2016.07.19 15:40:38 CST	2016.07.19 15:42:42 CST	Job_1467969296998_6856	Prediction Step1	2016st21	root.default	SUCCEEDED	96	96	8

根据Job ID链接进入Job详细页面，如下所示：

Counters for job_1467969296998_6856										
Counter Group		Counters								
		Name	Map	Reduce	Total					
File System Counters	FILE: Number of bytes read	0	388,604	388,604						
	FILE: Number of bytes written	11,526,274	1,314,572	12,840,846						
	FILE: Number of large read operations	0	0	0						
	FILE: Number of read operations	0	0	0						
	FILE: Number of write operations	0	0	0						
	HDFS: Number of bytes read	12,052,567,186	0	12,052,567,186						
	HDFS: Number of bytes written	0	21,549	21,549						
	HDFS: Number of large read operations	0	0	0						
	HDFS: Number of read operations	288	24	312						
	HDFS: Number of write operations	0	16	16						
Job Counters	Data-local map tasks	0	0	78						
	Launched map tasks	0	0	96						
	Launched reduce tasks	0	0	8						
	Rack-local map tasks	0	0	18						
	Total megabyte-seconds taken by all map tasks	0	0	1,560,350,720						
	Total megabyte-seconds taken by all reduce tasks	0	0	732,163,072						
	Total time spent by all map tasks (ms)	0	0	1,523,780						
	Total time spent by all maps in occupied slots (ms)	0	0	1,523,780						
	Total time spent by all reduce tasks (ms)	0	0	715,003						
	Total time spent by all reduces in occupied slots (ms)	0	0	715,003						
Map-Reduce Framework	Total vcore-seconds taken by all map tasks	0	0	1,523,780						
	Total vcore-seconds taken by all reduce tasks	0	0	715,003						
	Combine input records	127,821,389	0	127,821,389						
	Combine output records	9,821	0	9,821						
	CPU time spent (ms)	1,425,080	36,170	1,461,250						
	Failed Shuffles	0	0	0						
	GC time elapsed (ms)	59,655	949	60,604						
	Input split bytes	11,424	0	11,424						
	Map input records	128,713,807	0	128,713,807						
	Map output bytes	4,081,017,223	0	4,081,017,223						
Shuffle Errors	Map output materialized bytes	393,164	0	393,164						
	Map output records	127,821,389	0	127,821,389						
	Merged Map outputs	0	768	768						
	Physical memory (bytes) snapshot	25,785,630,720	1,345,679,360	27,131,310,080						
	Reduce input groups	0	8,282	8,282						
	Reduce input records	0	9,821	9,821						
	Reduce output records	0	560	560						
	Reduce shuffle bytes	0	393,164	393,164						
	Shuffled Maps	0	768	768						
	Spilled Records	9,821	9,821	19,642						
File Input Format Counters	Total committed heap usage (bytes)	20,063,453,184	1,536,688,128	21,600,141,312						
	Virtual memory (bytes) snapshot	157,729,685,504	13,223,530,496	170,953,216,000						
File Output Format Counters	Name	Map	Reduce	Total						
	Bytes Written	0	21,549	21,549						

2. Step 2

(1) 集群All Application (<http://114.212.190.91:8088/>) WebUI页面中查看Job的执行状态

Scheduler Metrics										
Scheduler Type	Scheduling Resource Type		Minimum Allocation			Maximum Allocation				
Fair Scheduler	[MEMORY, CPU]		<memory:1024, vCores:1>			<memory:8192, vCores:8>				
Show 100 ▾ entries										
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1467969296998_6859	2016st21	Prediction Step2	MAPREDUCE	root.default	Tue Jul 19 15:43:22 +0800 2016	Tue Jul 19 15:43:47 +0800 2016	FINISHED	SUCCEEDED		History
application_1467969296998_6856	2016st21	Prediction Step1	MAPREDUCE	root.default	Tue Jul 19 15:40:30 +0800 2016	Tue Jul 19 15:42:43 +0800 2016	FINISHED	SUCCEEDED		History

(2) WebUIx页面 (<http://114.212.190.91:19888/jobhistory>)

Show 100 ▾ entries										
Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total
2016.07.19 15:43:22 CST	2016.07.19 15:43:29 CST	2016.07.19 15:43:47 CST	job_1467969296998_6859	Prediction Step2	2016st21	root.default	SUCCEEDED	8	8	1

根据Job ID链接进入Job详细页面，如下所示：

loop Counters for job_1467969296998_6859											
Counter Group		Counters									
		Name	Map	Reduce	Total						
File System Counters	FILE: Number of bytes read	0	22,797		22,797						
	FILE: Number of bytes written	949,311	138,551		1,087,862						
	FILE: Number of large read operations	0	0		0						
	FILE: Number of read operations	0	0		0						
	FILE: Number of write operations	0	0		0						
	HDFS: Number of bytes read	22,533	0		22,533						
	HDFS: Number of bytes written	0	10		10						
	HDFS: Number of large read operations	0	0		0						
	HDFS: Number of read operations	24	3		27						
	HDFS: Number of write operations	0	2		2						
Job Counters	Data-local map tasks	0	0		6						
	Launched map tasks	0	0		8						
	Launched reduce tasks	0	0		1						
	Rack-local map tasks	0	0		2						
	Total megabyte-seconds taken by all map tasks	0	0		43,379,712						
	Total megabyte-seconds taken by all reduce tasks	0	0		9,066,496						
	Total time spent by all map tasks (ms)	0	0		42,363						
	Total time spent by all maps in occupied slots (ms)	0	0		42,363						
	Total time spent by all reduce tasks (ms)	0	0		8,854						
	Total time spent by all reduces in occupied slots (ms)	0	0		8,854						
Map-Reduce Framework	Total vcore-seconds taken by all map tasks	0	0		42,363						
	Total vcore-seconds taken by all reduce tasks	0	0		8,854						
	Combine input records	560	0		560						
	Combine output records	560	0		560						
	CPU time spent (ms)	5,150	1,990		7,140						
	Failed Shuffles	0	0		0						
	GC time elapsed (ms)	405	95		500						
	Input split bytes	984	0		984						
	Map input records	560	0		560						
	Map output bytes	21,671	0		21,671						
Shuffle Errors	Map output materialized bytes	22,839	0		22,839						
	Map output records	560	0		560						
	Merged Map outputs	0	8		8						
	Physical memory (bytes) snapshot	2,142,326,784	174,460,928		2,316,787,712						
	Reduce input groups	0	560		560						
	Reduce input records	0	560		560						
	Reduce output records	0	1		1						
	Reduce shuffle bytes	0	22,839		22,839						
	Shuffled Maps	0	8		8						
	Spilled Records	560	560		1,120						
File Input Format Counters	Total committed heap usage (bytes)	1,610,612,736	201,326,592		1,811,939,328						
	Bytes Read	13,166,526,464	1,653,231,616		14,819,758,080						
	Bytes Written	0	10		10						

参 考 文 献

- [1] 黄宣华. 深入理解大数据 大数据处理与编程实践[M]. 北京: 机械工业出版社, 2014.7.
- [2] 日志分析方法概述- 百度技术博客- 51CTO技术博客
<http://baidutech.blog.51cto.com/4114344/743786/>
- [3] 海量Web日志分析用Hadoop提取KPI统计指标— 粉丝日志
<http://blog.fens.me/hadoop-mapreduce-log-kpi/>
- [4] 赵龙. 基于hadoop的海量搜索日志分析平台的设计和实现[D]. 大连理工大学, 2013.