

大数据实验4—图的三角形计数 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)

摘 要： 本次实验我们小组通过课堂上所讲的Hadoop的网页排名图算法PageRank的相关内容的启发，完成了社交网络局部关系图的三角形计数任务。并在集群上使用Twitter数据集与Google +数据集分别进行三角形的数目统计任务。并完成了选作任务转换逻辑的替换工作，在集群上执行得到相应的结果。

关键词： Hadoop、有向图、社交网络、局部关系图

§1. 引 言

一个社交网络可以看做是一张图。社交网络中的人对应于图的顶点；社交网络中的人际关系对应于图中的边。由一个社交网络抽象转换所成的无向图上便肯定存在大量的三角形。本次实验中，我们小组完成了社交网络局部关系图的三角形计数任务：(1)根据不同的逻辑(OR / AND)将有向图转换为无向图，(2)在无向图的基础上计算出三角形的个数，(3)并在集群上使用Twitter数据集与Google +数据集分别进行三角形的数目统计任务。

实验报告的第2节简要介绍实验环境和完成情况。第3节中将详细介绍实验各个部分的设计。测试与运行的结果留在第4节中展示。在第5节中总结实验内容和团队合作。

*陈越琦：121160005 Yueqichen.0x0@gmail.com 刘威：131220085 liuwei13cs@smail.nju.edu.cn
杨杰才：121160005 mark_grove@qq.com

§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

本次实验，我们分别完成了以下工作：

1. 分别根据OR 和AND 逻辑将输入文件中的有向图转换为无向图.
2. 统计图中已存在的边和构造三角形所需的边，并根据统计结果计算出无向图中存在的三角形个数.
3. 使用Driver程序将多个job合并.
4. 分别在Twitter 和Google+ 数据集完成测试。

§3. 实验具体设计

3.1 必做部分（OR逻辑转换）

3.1.1 总体设计

为完成在OR逻辑转换下的图的三角形计数，我们总共设计了两个MapReduce程序顺序执行来完成这一任务：

- DigraphToUngraph: 读入输入文件根据转换逻辑构造无向边，并统计出每一个顶点的相邻顶点并输出到HDFS.
- InNeed: 统计图中已有的边与构造三角形所需边，根据所需边与已有边对应关系统计三角形数目并输出.

DigraphToUngraph :

为完成DigraphToUngraph任务我们共设计了个3类：

1. DigraphToUngraphMapper类：分析输入文件的每一行，切割出两个顶点的ID，选择小的ID作为key, 大的ID作为value.

2. DigraphToUngraphReducer类：将key相同的ID所对应的value结合起来作为输出value, 获得该ID所代表的顶点的所有相邻顶点的ID,并输出到HDFS.
3. DigraphToUngraph类:主类，负责启动配置作业，提交作业与获得完成结果的过程.

DigraphToUngraph的MapReduce各阶段的K,V类型见图2:

		Map	Reduce
输入	key	当前行的offset	较小的节点ID
	value	行中内容	较大的节点ID
输出	key	较小的节点ID	节点ID
	value	较大的节点ID	与key相邻的节点的ID

图 2 K,V类型表

InNeed :

为完成InNeed任务我们共设计了3个类:

1. InNeedMapper类：分析DigraphToUngraph任务的输出文件，获得顶点ID与相邻顶点ID的对应关系，统计出图中已有的无向边，以及要构成三角形需存在的边（具体判断方法：假设key为点A，从上一步结果得知其有两个相邻顶点B与C，则边BC就是构成三角形需要的边）。将边的两个顶点ID的组合作为key，value为边的类型标记，存在的边标记为&，需要的边标记为#。
2. InNeedReducer类：查询Map的输出，如果边是已存在边，则记录其存在状态。如果类型是构成三角形需存在的边，且该边已经被标记为存在，则说明三角形存在，三角形数目加1。最后在cleanup函数中输出最终的三角形数目统计结果。
3. InNeed类：主类，负责启动配置作业，提交作业与获得完成结果的过程.

InNeed的MapReduce各阶段的K,V类型见图3:

		Map	Reduce
输入	key	节点ID	边的两端点序列
	value	与key节点相邻的节点的ID	边的类型 (& / #)
输出	key	边的两端点序列	边的两端点序列
	value	边的类型 (& / #)	边的类型 (& / #)

图 3 K,V类型表

3.1.2 DigraphToUngraph任务的伪代码

DigraphToUngraphMapper:

Class 1 *class Mapper*

Input : 输入文件

Output : 键值对 (顶点ID, 顶点ID)

```

1: procedure MAP(Text t)
2:   vertex1_ID  $\leftarrow$  t.split
3:   vertex2_ID  $\leftarrow$  t.split
4:   if vertex1_ID < vertex2_ID then
5:     Emit((vertex1_ID, vertex2_ID))
6:   else
7:     Emit((vertex2_ID, vertex1_ID))
8:   end if
9: end procedure

```

DigraphToUngraphReducer:

Class 2 *class Reducer*

Input : 键值对 (顶点ID, 顶点ID)

Output : 顶点a所代表ID 顶点a的相邻顶点所代表ID的序列

```

1: procedure REDUCE(key: vertex_smaller_ID, value: vertex_greater_ID)
2:   out  $\leftarrow$   $\emptyset$ 
3:   for value in values do
4:     out  $\leftarrow$  out + value: vertex_greater_ID
5:   end for
6:   Write(key: vertex_smaller_ID, out)
7: end procedure

```

3.1.3 InNeed任务的伪代码

InNeedMapper:

Class 3 *class Mapper*

Input : 输入文件: 顶点ID 所有相邻顶点ID

Output : 键值对 (边的ID序列, 边的种类标记)

```

1: procedure MAP(Key keyID, Value value_IDS)
2:   IDs  $\leftarrow$  value_IDS.split
3:   for ID in IDs do
4:     Emit((keyID#ID, &))
5:   end for
6:   for ID1 in IDs do
7:     for ID2 in IDs do
8:       Emit((ID1#ID2, #))
9:     end for
10:  end for
11: end procedure

```

InNeedReducer:

Class 4 *class Reducer*

Input : 键值对 (边的ID序列, 边的种类标记)

Output : 该Reduce统计出的三角形数目

```

1: procedure REDUCE(key: ID1#ID2, values: # or &))
2:   NumTran  $\leftarrow$  0
3:   count  $\leftarrow$  0
4:   Exist  $\leftarrow$  false
5:   for value in values do
6:     if value == & then
7:       Exist  $\leftarrow$  true
8:     end if
9:     if value == # then
10:      count  $\leftarrow$  count + 1
11:    end if
12:  end for
13:  if Exist == true then
14:    NumTran  $\leftarrow$  NumTran + count
15:  end if
16: end procedure

```

3.2 选做部分（AND逻辑转换）

3.2.1 总体设计

当逻辑变为IF $(A \rightarrow B)$ AND $(B \rightarrow A)$ THEN A-B时，我们考虑首先对图的边进行检查，若两个方向的边均存在，则说明该边存在。为做到这一点，我们考虑首先从输入数据中去除只有一个方向的边，经过这一处理后，便可以用必做部分的两个任务依次执行来完成这一任务。因而我们编写了StrongCheck这一任务，该任务读取输入数据，输出两个方向均存在边的顶点ID对的编号（按照顶点ID从小到大的顺序）。为实现该任务，我们采用了类似于前面InNeed任务的思路，读入一对顶点ID (A,B)，将顶点ID对(A,B)设为存在，其反向(B,A)设为需要，从而在Reduce时检查(B,A)是否存在。通过以上方法即可将单方向的边去除。最后，将StrongCheck的输出文件作为输入依次执行必做部分所述的两个任务，即可完成欲执行的任务。

StrongCheck任务的具体设计如下：

为完成StrongCheck任务我们共设计了3个类：

1. StrongCheckMapper类：分析输入的txt文件，依次读取一对顶点ID，将两个顶点ID的组合作为key，正向组合作为存在边，反向组合作为需求边。value为边的类型标记，存在的边标记为&，需要的边标记为#。
2. StrongCheckReducer类：查询Map的输出，如果边是已存在边，则记录其存在状态。如果边是需求边，则记录其需求状态。若一条边既是存在边也是需求边，则说明这一条边是双向的，应该转换为无向边，此时按照从小大顺序输出这一对顶点ID。
3. StrongCheck类：主类，负责启动配置作业，提交作业与获得完成结果的过程。

StrongCheck的MapReduce各阶段的K,V类型见图3：

		Map	Reduce
输入	key	当前行的offset	顶点ID对
	value	每一行的顶点ID对	边的类型 (&/#)
输出	key	顶点ID对	转换后无向边较小的节点ID
	value	边的类型 (&/#)	转换后无向边较大的节点ID

图 4 K,V类型表

3.2.2 StrongCheck任务的伪代码

StrongCheckMapper:

Class 5 *class Mapper*

Input : 输入文件: 顶点ID对

Output : 键值对 (边的ID序列, 边的种类标记)

```

1: procedure MAP(Key ID1, ID2)
2:   Emit((ID1#ID2, #))
3:   Emit((ID2#ID1, &))
4: end procedure

```

StrongCheckReducer:

Class 6 *class Reducer*

Input : 键值对 (边的ID序列, 边的种类标记)

Output : 键值对 (较小顶点ID, 较大顶点ID)

```

1: procedure REDUCE(key: ID1#ID2, values: # or &))
2:   Need  $\leftarrow$  false
3:   Exist  $\leftarrow$  false
4:   for value in values do
5:     if value == & then
6:       Exist  $\leftarrow$  true
7:     end if
8:     if value == # then
9:       Need  $\leftarrow$  true
10:    end if
11:  end for
12:  if Exist == true AND Need == true then
13:    vertex_smaller_ID, vertex_greater_ID  $\leftarrow$  key.split
14:    Emit((vertex_smaller_ID, vertex_greater_ID))
15:  end if
16: end procedure

```

3.3 对于Google+数据集所做的改进

在Twitter数据集上运行时, 我们采用了long来记录顶点的ID, 这样也可以直接比较顶点ID的大小。而对于Google+数据集, 很多顶点ID非常长, 转换成数字远超long所能表示的范围, 因而我们选择了采用String直接记录顶点ID对应的字符串, 而不是转换为响应的整数表示。为了比较两个顶点ID, 我们使用java的compareTo方法来比较两个顶点ID字符串, 这时比较的不是数字大小, 而是依次比较各个字符的ASCII码的值。

此外，对于OR逻辑的程序在Google+数据集上运行，生成无向图的边非常多，使得运行时间大幅增加。在这里为了提升速度，我们给InNeed任务设置了8个Reduce任务，这样在Reduce过程中速度可以明显提升。不过此时，三角形数目将分成八个部分统计，最后需要将这八个数据加在一起获得总的三角形数目。

§4. 实验运行结果与性能分析

4.1 JAR包执行方式说明

本次实验我们共编译并提交了三个JAR包：Driver1.jar Driver_op.jar Driver_G.jar。Driver1.jar用于在Twitter数据集上运行OR逻辑转换的程序，Driver_G.jar用于在Google+数据集上运行OR逻辑转换的程序，Driver_op.jar可用于运行Twitter与Google+两个数据集上AND逻辑转换的程序。这三个jar包的执行方式如下：

1. 首先使用scp指令将jar包传送到服务器并使用SSH登录到服务器
2. 使用以下指令执行OR逻辑程序在Twitter数据集上运行的Driver：`hadoop jar Driver1.jar Driver hdfs://master01:9000/data/graphTriangleCount/twitter_graph_v2.txt ./Lab4/output1 ./Lab4/output2`
3. 使用以下指令执行OR逻辑程序在Google+数据集上运行的Driver：`hadoop jar Driver_G.jar Driver hdfs://master01:9000/data/graphTriangleCount/gplus_combined.unique.txt ./Lab4/gp_output1 ./Lab4/gp_output2`
4. 使用以下指令执行AND逻辑程序在Twitter数据集上运行的Driver：`hadoop jar Driver_op.jar Driver hdfs://master01:9000/data/graphTriangleCount/twitter_graph_v2.txt ./Lab4/op_output1 ./Lab4/op_output2 ./Lab4/op_output3`
5. 使用以下指令执行AND逻辑程序在Google+数据集上运行的Driver：`hadoop jar Driver_op.jar Driver hdfs://master01:9000/data/graphTriangleCount/gplus_combined.unique.txt ./Lab4/op_gpoutput1 ./Lab4/op_gpoutput2 ./Lab4/op_gpoutput3`

4.2 实验运行结果展示

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。

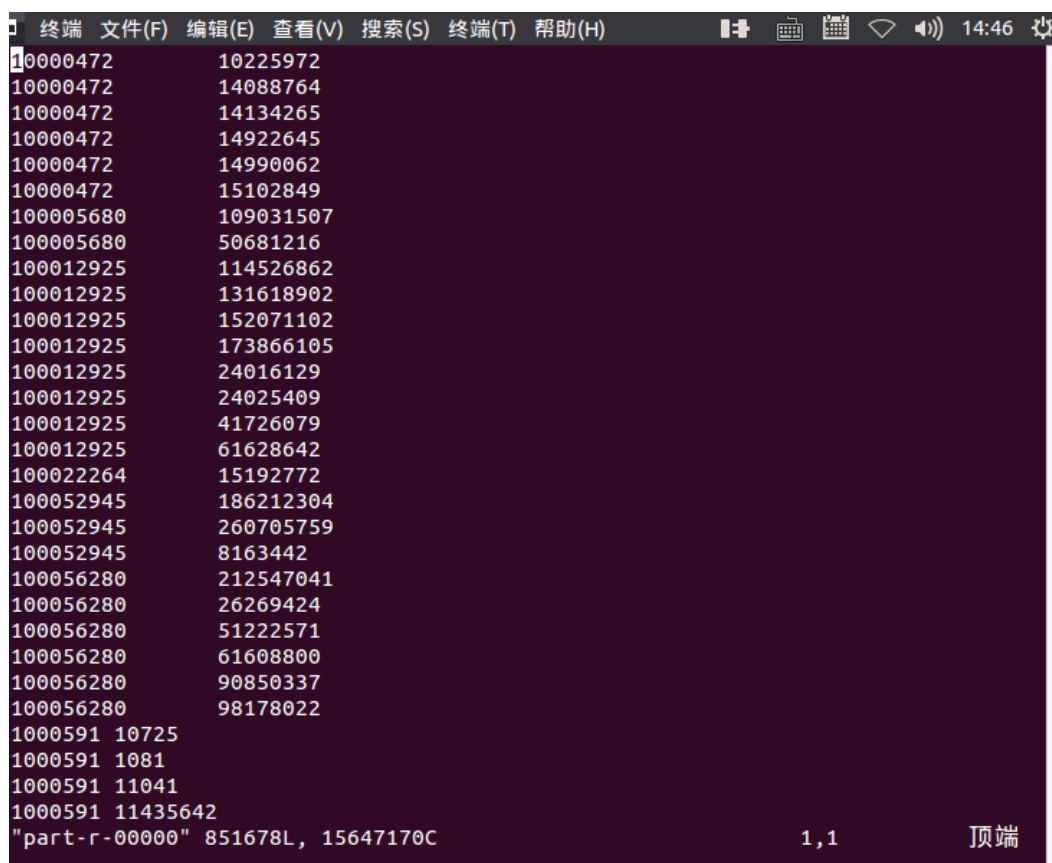
在集群上运行的结果如下：

各个任务统计出的三角形个数与运行时间见下面两表（由于集群运行速度波动较大，我们将任务执行多次后选择最短的时间，同时所有时间均未计入任务的排队时间）：

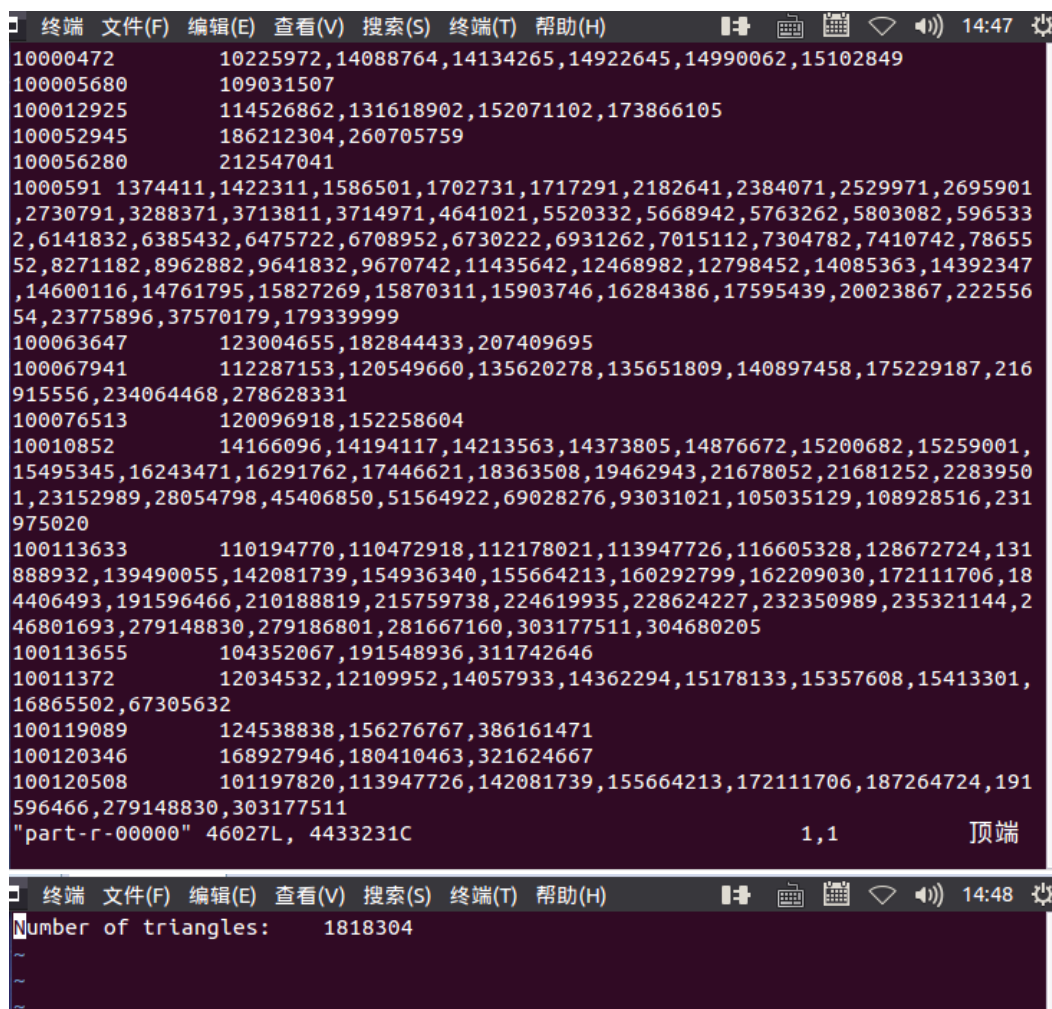
数据集	三角形个数	Driver程序在集群上的运行时间（秒）
Twitter	13082506	563
Google+	1073677742	28733

图 5 OR逻辑程序

- AND逻辑程序在Twitter数据集上运行的运行结果：该任务的输出结果分为三部分，分别为StrongCheck, DigraphToUngraph与InNeed的运行结果，其在HDFS上的存放路径分别为：hdfs://master01:9000/user/2016st21/Lab4/op_output1, hdfs://master01:9000/user/2016st21/Lab4/op_output2 与hdfs://master01:9000/user/2016st21/Lab4/op_output3 . 其中最后一个存放最后统计的三角形的结果
该任务在集群上运行的结果的部分截图如下



```
终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H) 14:46
10000472      10225972
10000472      14088764
10000472      14134265
10000472      14922645
10000472      14990062
10000472      15102849
100005680     109031507
100005680     50681216
100012925     114526862
100012925     131618902
100012925     152071102
100012925     173866105
100012925     24016129
100012925     24025409
100012925     41726079
100012925     61628642
100022264     15192772
100052945     186212304
100052945     260705759
100052945     8163442
100056280     212547041
100056280     26269424
100056280     51222571
100056280     61608800
100056280     90850337
100056280     98178022
1000591 10725
1000591 1081
1000591 11041
1000591 11435642
"part-r-00000" 851678L, 15647170C 1,1 顶端
```



```

终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H) 14:47
10000472      10225972,14088764,14134265,14922645,14990062,15102849
100005680      109031507
100012925      114526862,131618902,152071102,173866105
100052945      186212304,260705759
100056280      212547041
1000591 1374411,1422311,1586501,1702731,1717291,2182641,2384071,2529971,2695901
,2730791,3288371,3713811,3714971,4641021,5520332,5668942,5763262,5803082,596533
2,6141832,6385432,6475722,6708952,6730222,6931262,7015112,7304782,7410742,78655
52,8271182,8962882,9641832,9670742,11435642,12468982,12798452,14085363,14392347
,14600116,14761795,15827269,15870311,15903746,16284386,17595439,20023867,222556
54,23775896,37570179,179339999
100063647      123004655,182844433,207409695
100067941      112287153,120549660,135620278,135651809,140897458,175229187,216
915556,234064468,278628331
100076513      120096918,152258604
10010852      14166096,14194117,14213563,14373805,14876672,15200682,15259001,
15495345,16243471,16291762,17446621,18363508,19462943,21678052,21681252,2283950
1,23152989,28054798,45406850,51564922,69028276,93031021,105035129,108928516,231
975020
100113633      110194770,110472918,112178021,113947726,116605328,128672724,131
888932,139490055,142081739,154936340,155664213,160292799,162209030,172111706,18
4406493,191596466,210188819,215759738,224619935,228624227,232350989,235321144,2
46801693,279148830,279186801,281667160,303177511,304680205
100113655      104352067,191548936,311742646
10011372      12034532,12109952,14057933,14362294,15178133,15357608,15413301,
16865502,67305632
100119089      124538838,156276767,386161471
100120346      168927946,180410463,321624667
100120508      101197820,113947726,142081739,155664213,172111706,187264724,191
596466,279148830,303177511
"part-r-00000" 46027L, 4433231C      1,1      顶端

终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H) 14:48
Number of triangles: 1818304
~
~
~

```

可以看到，最后统计出的三角形数目为1818304，明显少于上一结果。

- OR逻辑程序在Google+数据集上运行的运行结果：该任务的输出结果分为两部分，分别为DigraphToUngraph与InNeed的运行结果，其在HDFS上的存放路径分别为：
hdfs://master01:9000/user/2016st21/Lab4/gp_output1 与 hdfs://master01:9000/user/2016st21/Lab4/gp_output2 . 其中后者存放最后统计的三角形的结果
该任务在集群上运行的结果的部分截图如下

```

00007793076838098007 100011386455690017279,100066493310937980219,100082296680
929174350,100091126610411008053,100095264391620936473,100095669250946685587,1000
97237485500025852,100099806002888081826,100119377012725467053,100124389520659251
191,100149175673005374822,100154520568840975178,100173452066172931939,1001776650
92199474358,100192309543043196568,100246857434952148531,100262595546646927505,10
0272572056305744950,100300281975626912157,100382076211124259598,1003835268056512
07186,100397511207083609950,100414409026788672356,100458071620246434265,10048294
2881898107570,100483930704326945631,100512649718649402368,100518419853963396365,
100523784851251213675,10053538638690515335,100612175927429294541,10062253164605
9523513,100715738096376666180,100771715351072186974,100834378485895409468,100861
100317455495250,100894513529515310753,100933312428715463057,10094071689231372728
5,100964406424233231915,101005240119432390037,101035196437264488455,101063636930
597622082,101101582632461364394,101110424233338407044,101174714442258565659,1011
74951617223562800,101215103890643218195,101235095970404720691,101236358663985370
139,101256198205956792432,101261243957067319422,101425605470782133226,1014749649
11075679569,101479626695467843303,101483533411566453214,101484760208373359735,10
1567641239571518754,101629211371073711149,101638237926824161827,1016498928390360
81330,101697775213251991950,101704103161442695877,101737255974887534884,10175637
8878167883912,101766673994203254909,101793532287583914396,101817732309915049788,
101849747879612982297,101856499111827242425,101879656850551340719,10191522629599
6051057,101962829696981365403,101965795444195512315,101996370517328694418,102034
052532213921839,102048265612444661933,102063241396469400293,10207520133376960044
7,102148161536515462447,102170431816592344972,102176881069394102707,102178700954
1,1 Top

```

为加速Reduce过程，我们为InNeed作业设置了8个Reduce任务，将8个任务各自统计出的三角形数目相加，统计出总的三角形数目为1073677742。

- AND逻辑程序在Google+数据集上运行的运行结果：该任务的输出结果分为三部分，分别为StrongCheck，DigraphToUngraph与InNeed的运行结果，其在HDFS上的存放路径分别为：`hdfs://master01:9000/user/2016st21/Lab4/op_gpoutput1`，`hdfs://master01:9000/user`

/2016st21/Lab4/op_gpoutput2 与hdfs://master01:9000/user/2016st21/Lab4/op_gpoutput3

. 其中最后一个存放最后统计的三角形的结果

该任务在集群上运行的结果的部分截图如下

```

100000486454408208159 100567638549968031422
100000763912458474062 101695373615640250123
100000763912458474062 107153364124481663469
100000763912458474062 114716671631005124422
100000763912458474062 114800747268559229378
100000763912458474062 114864017893355511120
100000763912458474062 117892288416322141592
100000763912458474062 117985061742250226747
100000772955143706751 100034852139736181861
100000772955143706751 100163594096411699235
100000772955143706751 100238683227109911259
100000772955143706751 100264010813126229552
100000772955143706751 100534424316001246594
100000772955143706751 100597038897915499206
100000772955143706751 100874515265830001658
100000772955143706751 100950274366779147873
100000772955143706751 101127105424399401348
100000772955143706751 101144838792283772783
100000772955143706751 101291405885251974009
100000772955143706751 101303101272501376461
100000772955143706751 101324614119590756927
100000772955143706751 101367799604091052665
100000772955143706751 101532329045297515791

22,1 Top

100000486454408208159 100567638549968031422
100000763912458474062 101695373615640250123,107153364124481663469,114716671631
005124422,114800747268559229378,114864017893355511120,117892288416322141592,1179
85061742250226747

Number of triangles: 27018510

```

可以看到，最后统计出的三角形数目为27018510。同样，该结果也明显少于上一个统计结果。

本实验在集群上执行MapReduce Job后获得的执行报告如下（由于本实验任务较多，我们只在文档中贴出OR逻辑程序在Twitter数据集上运行的任务的执行报告，其余Job的截图可在

执行报告文件夹中查看)：

- 在集群All Application (<http://114.212.190.91:8088/>) 的WebUI页面中查看Job的执行状态的截图如下：

Show 20 ▾ entries											Search: <input type="text"/>	
	ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	
2016.05.30 23:17:17 CST	2016.05.30 23:17:23 CST	2016.05.30 23:26:48 CST	job_1464281797944_1065	needed edge	2016st21	root.default	SUCCEEDED	8	8	1	1	
2016.05.30 23:13:33 CST	2016.05.30 23:16:27 CST	2016.05.30 23:17:09 CST	job_1464281797944_1062	Translation Di->Un	2016st21	root.default	SUCCEEDED	1	1	8	8	

- 在WebUI页面 (<http://114.212.190.91:19888/jobhistory>) 找到对应的job如下

Show 100▼ entries											Search: 1341	
Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total	Reduces Completed	
2016.05.30 23:13:33 CST	2016.05.30 23:16:27 CST	2016.05.30 23:17:09 CST	job_1464281797944_1062	Translation Di->Un	2016st21	root.default	SUCCEEDED	1	1	8	8	
2016.05.30 23:17:17 CST	2016.05.30 23:17:23 CST	2016.05.30 23:26:48 CST	job_1464281797944_1065	needed edge	2016st21	root.default	SUCCEEDED	8	8	1	1	

- 根据Job ID链接进入Job详细页面，几个job的详细信息如下所示。

– Translation Di–>Un:

Counters for job_1464281797944_1062					Logged in as: admin			
Counter Group	Name	Map	Reduce	Total				
File System Counters	HDFS: Number of write operations	0	16	16				
	HDFS: Number of read operations	3	24	27				
	HDFS: Number of large read operations	0	0	0				
	HDFS: Number of bytes written	0	12,820,282	12,820,282				
	HDFS: Number of bytes read	32,467,364	0	32,467,364				
	FILE: Number of write operations	0	0	0				
	FILE: Number of read operations	0	0	0				
	FILE: Number of large read operations	0	0	0				
	FILE: Number of bytes written	36,119,105	36,926,192	73,045,297				
	FILE: Number of bytes read	0	36,003,552	36,003,552				
Job Counters	Killed reduce tasks	0	0	1				
	Launched map tasks	0	0	1				
	Launched reduce tasks	0	0	9				
	Red-local map tasks	0	0	1				
	Total mapbyte-seconds taken by all map tasks	0	0	17,098,752				
	Total mapbyte-seconds taken by all reduce tasks	0	0	68,296,704				
	Total time spent by all map tasks (ms)	0	0	16,696				
	Total time spent by all maps in occupied slots (ms)	0	0	16,696				
	Total time spent by all reduce tasks (ms)	0	0	66,696				
	Total time spent by all reduces in occupied slots (ms)	0	0	66,696				
	Total vcore-seconds taken by all map tasks	0	0	16,696				
	Total vcore-seconds taken by all reduce tasks	0	0	66,696				
	Combine input records	0	0	0				
	Combine output records	0	0	0				
	CPU time spent (ms)	9,960	43,870	53,830				
	Failed Shuffles	0	0	0				
	GC time elapsed (ms)	191	1,561	1,752				
	Input split bytes	130	0	130				
	Map input records	1,768,135	0	1,768,135				

Map-Reduce Framework	Total vcores-seconds taken by all reduce tasks		0	0	66,696
	Name	Map	Reduce	Total	
	Combine input records	0	0	0	
	Combine output records	0	0	0	
	CPU time spent (ms)	9,960	43,870	53,830	
	Failed Shuffles	0	0	0	
	GC time elapsed (ms)	191	1,561	1,752	
	Input split bytes	130	0	130	
	Map input records	1,768,135	0	1,768,135	
	Map output bytes	32,467,234	0	32,467,234	
	Map output materialized bytes	36,003,552	0	36,003,552	
	Map output records	1,768,135	0	1,768,135	
	Merged Map outputs	0	8	8	
	Physical memory (bytes) snapshot	266,080,256	1,437,659,136	1,703,739,392	
	Reduce input groups	0	70,840	70,840	
	Reduce input records	0	1,768,135	1,768,135	
	Reduce output records	0	70,840	70,840	
	Reduce shuffle bytes	0	36,003,552	36,003,552	
	Shuffled Maps	0	8	8	
	Spilled Records	1,768,135	1,768,135	3,536,270	
	Total committed heap usage (bytes)	201,326,592	1,595,408,384	1,796,734,976	
Virtual memory (bytes) snapshot	1,645,809,664	13,224,067,072	14,869,876,736		
Shuffle Errors	Name	Map	Reduce	Total	
	BAD ID	0	0	0	
	CONNECTION	0	0	0	
	IO ERROR	0	0	0	
	WRONG LENGTH	0	0	0	
	WRONG MAP	0	0	0	
	WRONG REDUCE	0	0	0	
File Input Format Counters	Name	Map	Reduce	Total	
Bytes Read	32,467,234	0	32,467,234		
File Output Format Counters	Name	Map	Reduce	Total	
Bytes Written	0	12,820,282	12,820,282		

– needed edge

7

Counters for job_1464281797944_1065

logged job name

Counter Group	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	1,841,056,110	1,841,055,924	3,682,112,034
	FILE: Number of bytes written	3,683,034,754	1,841,171,209	5,524,205,963
	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	12,821,282	0	12,821,282
	HDFS: Number of bytes written	0	31	31
	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	Name	Map	Reduce	Total
	Data-local map tasks	0	0	7
	Launched map tasks	0	0	8
	Launched reduce tasks	0	0	1
	Red-Local map tasks	0	0	1
	Total map-seconds taken by all map tasks	0	0	424,556,544
	Total map-seconds taken by all reduce tasks	0	0	528,252,928
	Total time spent by all map tasks (ms)	0	0	414,606
	Total time spent by all maps in occupied slots (ms)	0	0	414,606
	Total time spent by all reduce tasks (ms)	0	0	515,872
	Total time spent by all reduces in occupied slots (ms)	0	0	515,872
	Total vcores-seconds taken by all map tasks	0	0	414,606
	Total vcores-seconds taken by all reduce tasks	0	0	515,872
	Name	Map	Reduce	Total
	Combine input records	0	0	0
	Combine output records	0	0	0
	CPU time spent (ms)	477,970	155,370	633,340
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	6,798	3,590	10,388
	Input split bytes	1,000	0	1,000
	Map input records	70,840	0	70,840
	Map output bytes	1,676,044,590	0	1,676,044,590
	Map output materialized bytes	0	0	0
Map-Reduce Framework	CPU time spent (ms)	477,970	155,370	633,340
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	6,798	3,590	10,388
	Input split bytes	1,000	0	1,000
	Map input records	70,840	0	70,840
	Map output bytes	1,676,044,590	0	1,676,044,590
	Map output materialized bytes	1,841,055,924	0	1,841,055,924
	Map output records	82,505,643	0	82,505,643
	Merged Map outputs	0	8	8
	Physical memory (bytes) snapshot	2,167,681,024	235,147,264	2,402,828,288
	Reduce input groups	0	39,031,397	39,031,397
	Reduce input records	0	82,505,643	82,505,643
	Reduce output records	0	1	1
	Reduce shuffle bytes	0	1,841,055,924	1,841,055,924
	Shuffled Maps	0	8	8
	Spilled Records	165,011,286	82,505,643	247,516,929
	Total committed heap usage (bytes)	1,672,478,720	170,917,888	1,843,396,608
	Virtual memory (bytes) snapshot	13,166,542,848	1,653,248,000	14,819,790,848
Shuffle Errors	Name	Map	Reduce	Total
	BAD ID	0	0	0
	CONNECTION	0	0	0
	IO ERROR	0	0	0
	WRONG LENGTH	0	0	0
	WRONG MAP	0	0	0
	WRONG REDUCE	0	0	0
File Input Format Counters	Name	Map	Reduce	Total
File Output Format Counters	Bytes Read	12,820,282	0	12,820,282
	Bytes Written	0	31	31

4.3 程序运行性能的分析

从上面的运行结果与运行时间可以看到，我们的程序主要的耗时之处在于InNeed任务，该任务的性能直接决定了运行时间的长短。在AND逻辑下，在第一个StrongCheck任务中，我们便消除了大量的单向边，这样使得InNeed任务执行时间大幅减少，从而也导致程序的运行时间有了非常明显的缩短。而对于OR逻辑，当数据集较大时，如果采用默认设置，性能会遇到一定的瓶颈，此时，我们通过使用`job.setNumReduceTasks`语句增加reduce任务的数目来提高速度。通过这种方法，程序的运行速度有了较为明显的提升。

4.4 程序性能及扩展性不足的分析

本次实验中，为了提高程序并行度与执行速度，可以设置第一个作业的reduce任务的数量，这样会使得第一个作业输出文件数量与reduce任务数相同。从而在Driver中的第二个作业中就可以提高mapper任务的数量，使得两次作业的并行度都能够得到提高，运行速度也有了十分明显的提升。但是由此带来了扩展性问题：如果在最后一个作业中设置了过多的reduce任务，最后得到统计结果的文件也会相应增加，要获得真正的总的三角形数量必须要人工将这些文件中的统计结果相加，由此程序扩展性就存在问题。这也是我们没有在最后一个作业中设置过多的reduce任务数目的原因，虽然这样能更进一步提升运行速度。

而对于选做任务，如果将前两个job合为一个job，直接判断双向的边是否均存在来进行有向图到无向图的转换而不是先筛除单向边再做转换，那么程序的性能还能得到进一步的提高。

此外，在作业执行过程中，我们发现存在作业执行严重的长尾延迟(Long Tail Latency)。在运行OR逻辑程序进行处理的几次过程中，多次出现由于slave3和slave17长时间处于高负荷状态，被分配到slave3和slave17上的任务执行明显落后于其他任务的执行，导致作业的执行被严重拖后的情况(此现象在运行于Google+数据集上表现的尤为明显，在这两个slave上的map与reduce任务执行时间比其他任务执行速度慢很多，导致运行时间多出数个小时。)。我们认为长尾延迟存在的原因可能是中间数据划分不均匀，或者是因为其他作业的存在导致集群负载不均衡。

§5. 实验总结

5.1 实验内容总结

本次实验是大数据处理综合实验平时实验的最后一次，在这次实验中我们完成了社交网络局部关系图的三角形计数任务：(1)根据不同的逻辑(OR / AND)将有向图转换为无向图，(2)在无向图的基础上计算出三角形的个数，(3)并在集群上使用Twitter数据集与Google +数据集分别进行三角形的数目统计任务。在实验中为了保持一致，图中所有的边都是以编号较小的点为key，编号较大的点(集)为value。

此外，针对不同的有向图无向图转换逻辑，本实验还分别编写了不同的Driver程序，实现多个MapReduce程序的批处理。更多关于程序运行性能和扩展性的分析在第4部分中详细叙述。因为本次实验的测试流程进行得比较晚，导致扎堆测试，浪费了很多的时间，给程序的测试带来了很大的麻烦。在以后的实验中应该吸取教训，尽早开始测试。

5.2 团队合作总结

本次实验中，大家分工完成必做部分和选做部分以及实验报告的撰写。有了前几次合作的经验，本次的合作很顺利。

参 考 文 献

- [1] 黄宜华. 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.