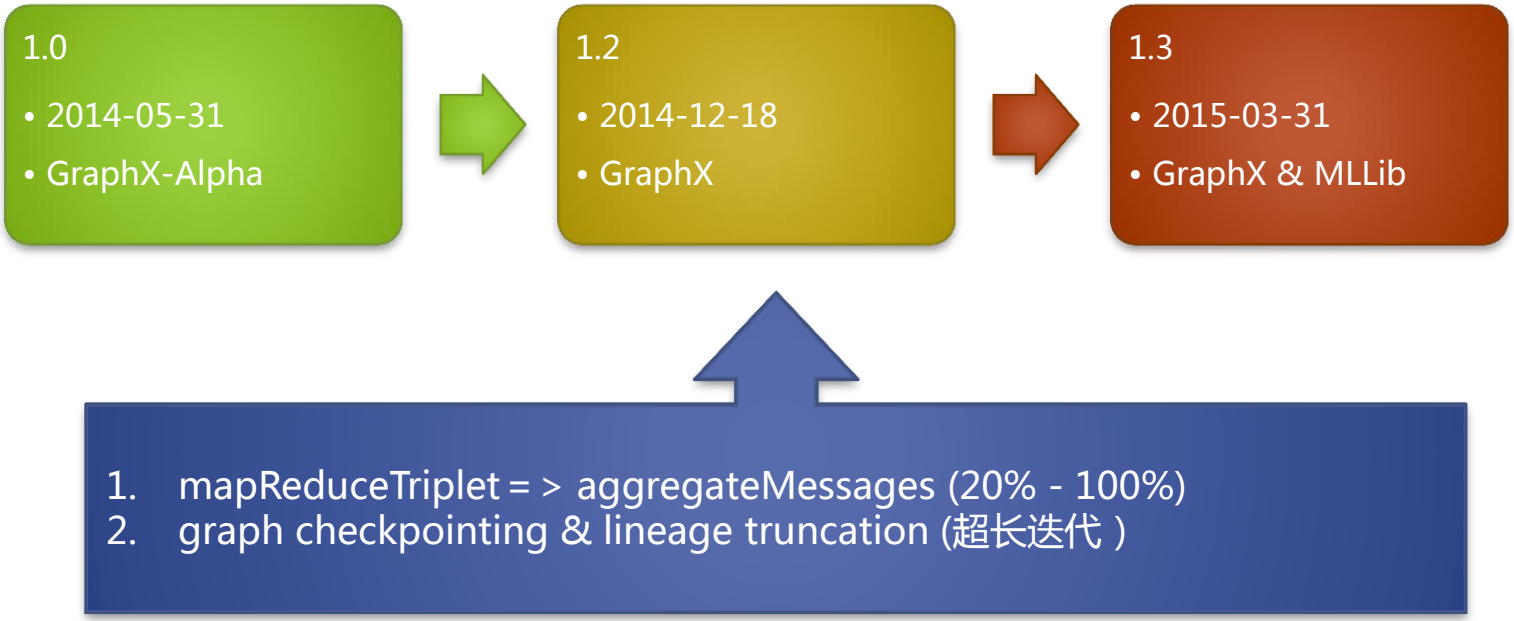
图流合璧

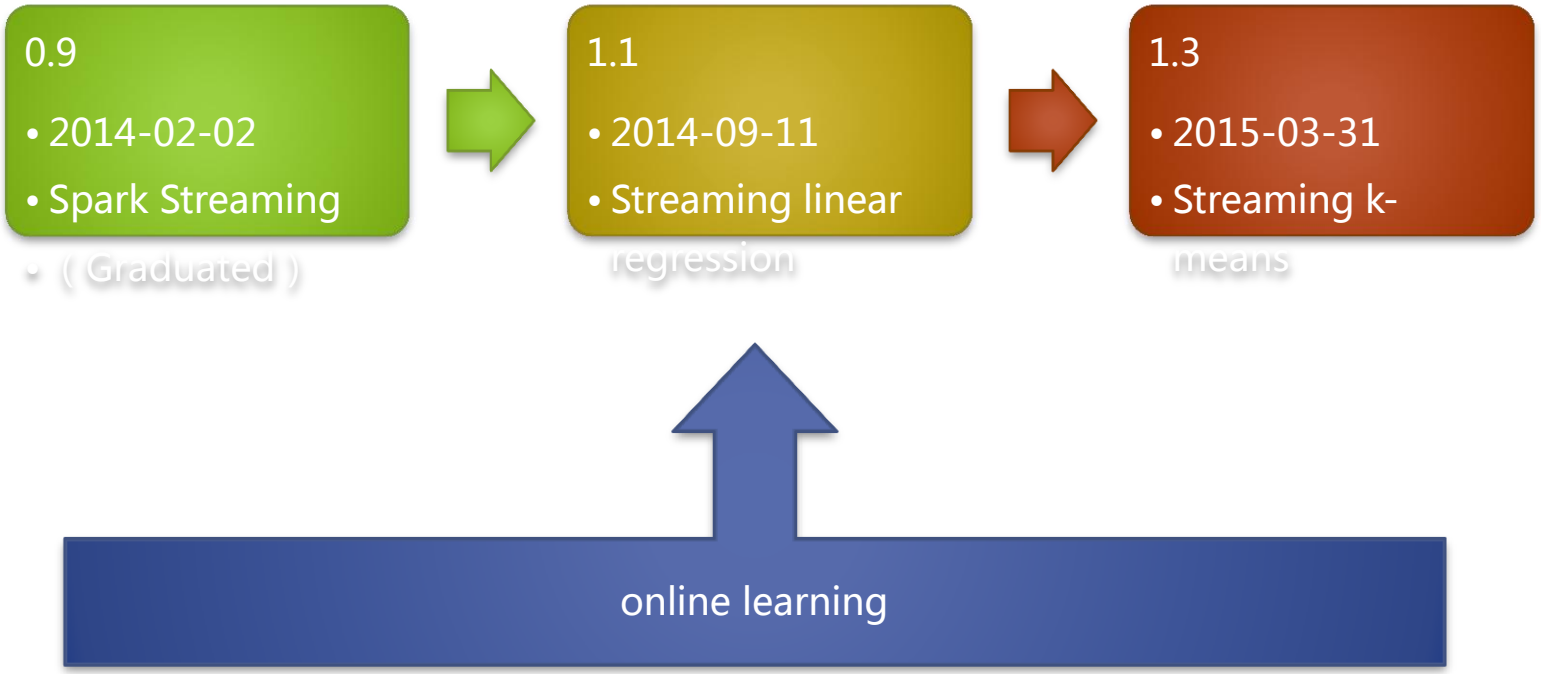
——基于Spark Streaming和GraphX的动态图计算

数据技术与产品部——数据科学
库里 衡云 明风

GraphX的回顾

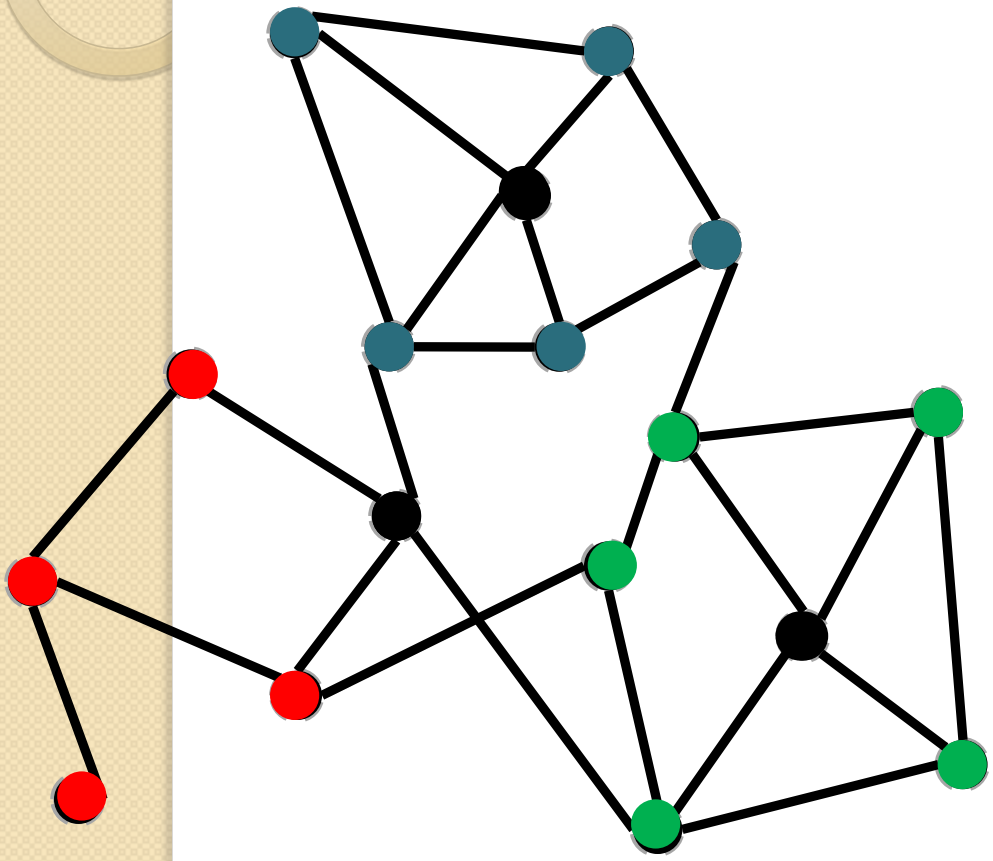

1.0
• 2014-05-31
• GraphX-Alpha
1.2
• 2014-12-18
• GraphX
1.3
• 2015-03-31
• GraphX & MLLib
1. mapReduceTriplet = > aggregateMessages (20% - 100%)
2. graph checkpointing & lineage truncation (超长迭代)

Streaming + MLlib 的发展



源起

杭州 Spark Meetup：《基于GraphX的社区发现》 刀剑　　2014-12-28



Louvain[1]算法

1、初始化，将每个节点划分在不同的社区中

2、对每个节点，计算Modularity [2]增益

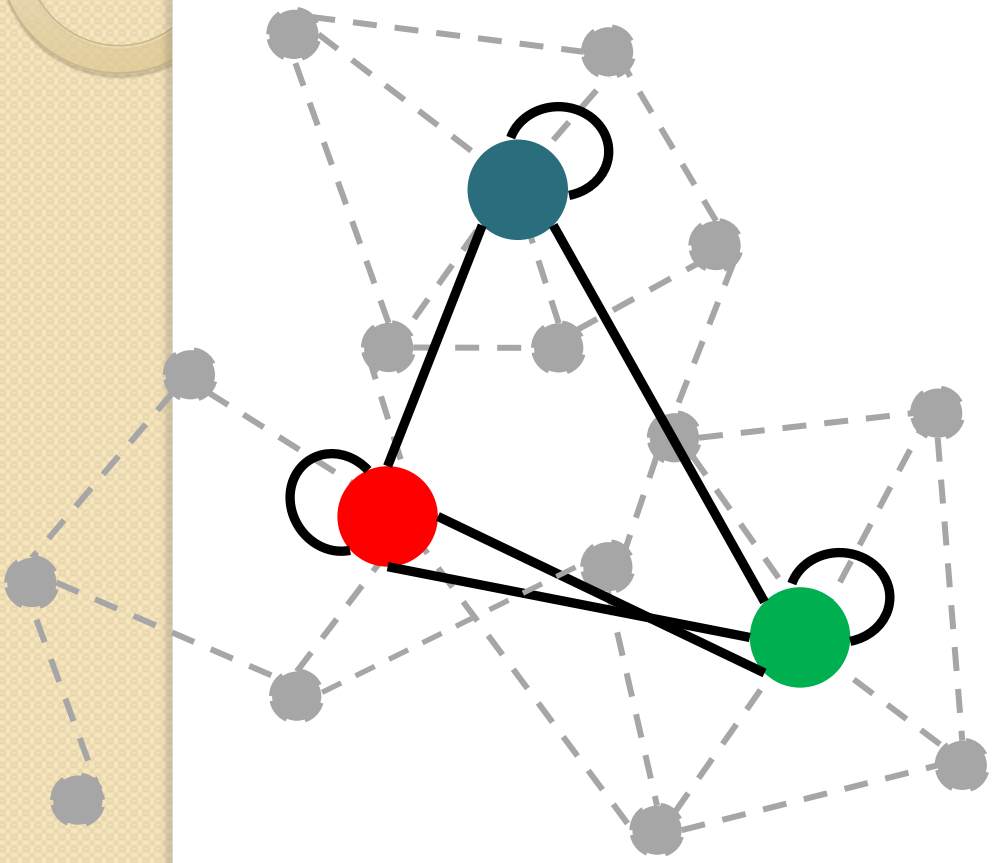
3、执行Unfolding，合并节点，减少节点数

4、构造新图

[1] Fast algorithm for detecting community structure in networks

源起

杭州 Spark Meetup：《基于GraphX的社区发现》 刀剑 2014-12-28



Louvain[1]算法

1、初始化，将每个节点划分在不同的社区中

2、对每个节点，计算Modularity [2]增益

3、执行Unfolding，合并节点，减少节点数

4、构造新图

Modularity的计算和意义

公式[2]：

$$Q = \frac{1}{2m}\sum_{i,j}[A_{ij} - \frac{k_i k_j}{2m}]\,\delta(c_i, c_j)$$

作用是衡量实际的图与随机构图之间的差异，表示对图划分的模块化程度。

上面公式需要对两两节点进行计算，为减少中间计算量可使用下面经推导后的公式：

$$Q = \sum_i^c [\frac{\sum \text{in}}{2m} - \left(\frac{\sum \text{tot}}{2m}\right)^2]$$

其中，表示**一个社区内部**的连线数，表示**一个社区**所有点的度数之和。

[2] Modularity and community structure in networks

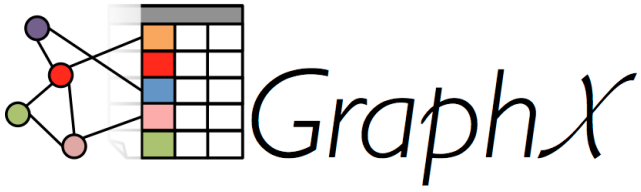
新的问题和挑战

- GraphX可以对用户进行社群划分，可以帮助更好地为决策提供依据，但是每天一次离线计算的结果不及时（2小时），没法快速响应业务

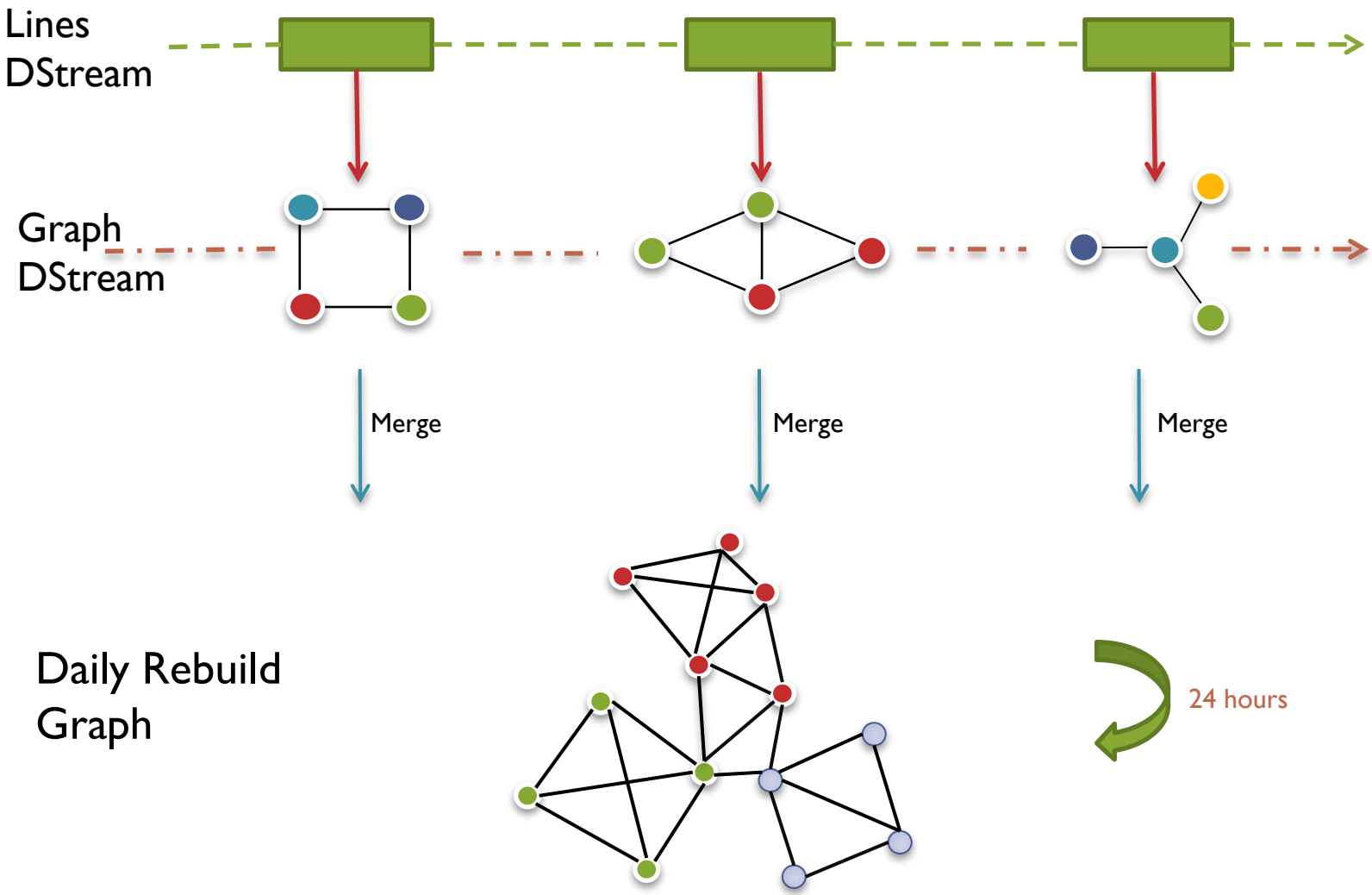
- 业务需要实时对淘宝的用户进行社群划分，实时观察到社群的演化过程，对淘宝的生态环境有更好的理解与掌握，做出更及时准确的决策

实时消息 + 实时图构建 = 动态图模型

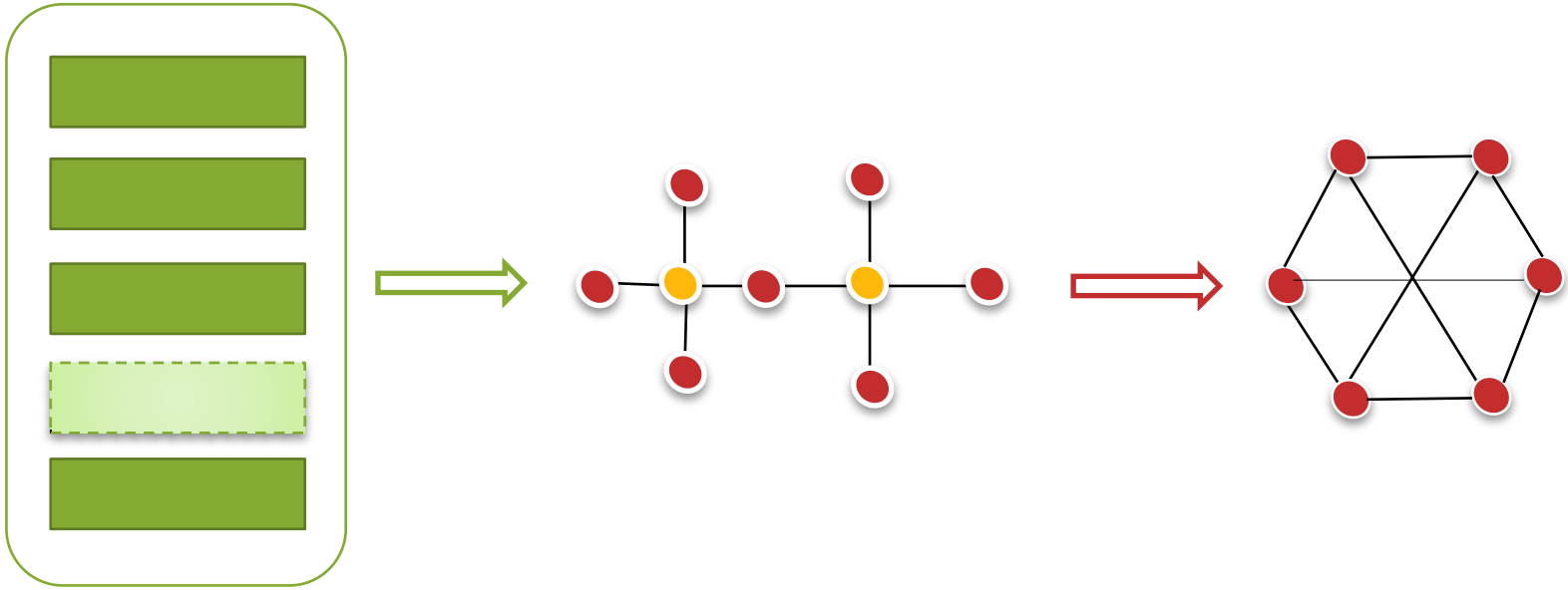
+



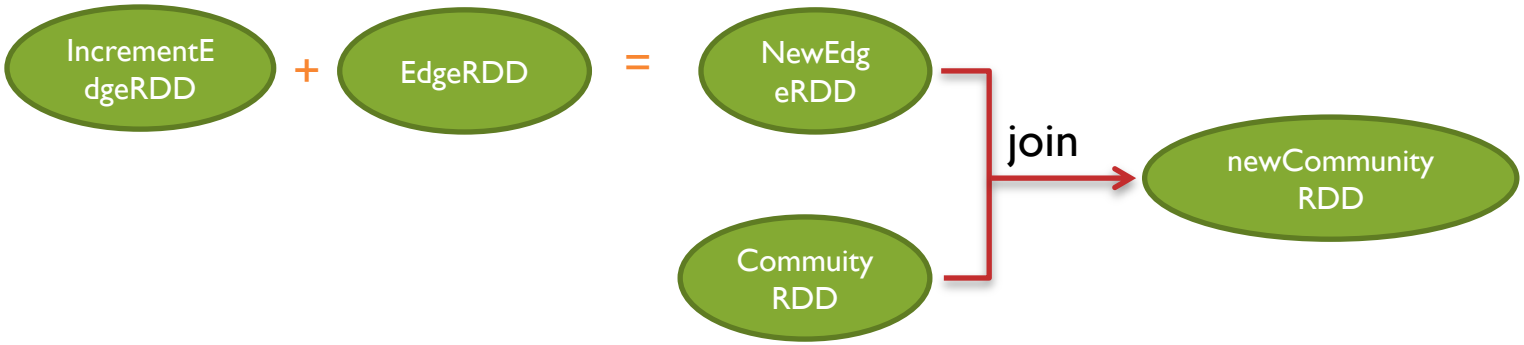
流式图计算

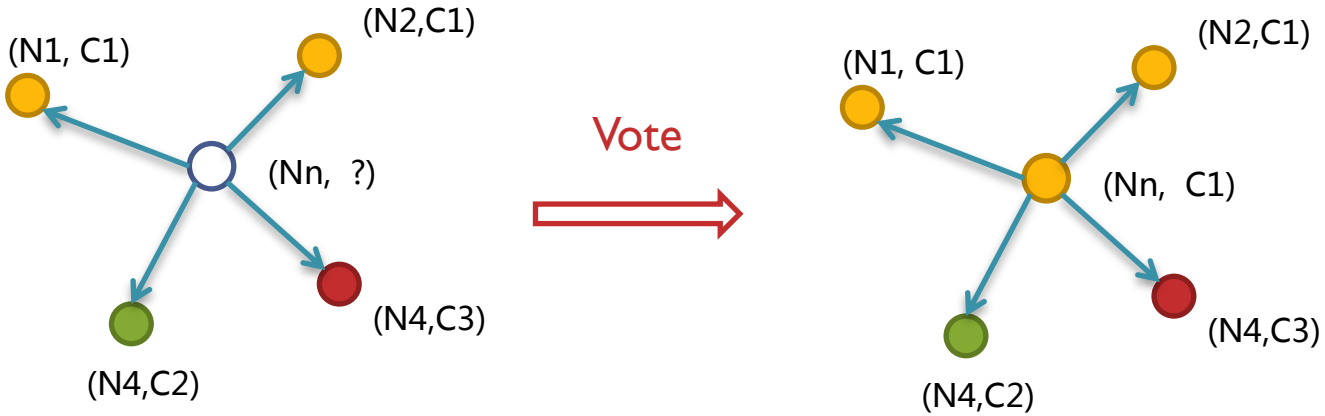

Lines
DStream
Graph
DStream
Merge
Merge
Merge
Daily Rebuild
Graph
24 hours

小图构建



融合模型1 (Join and Vote)





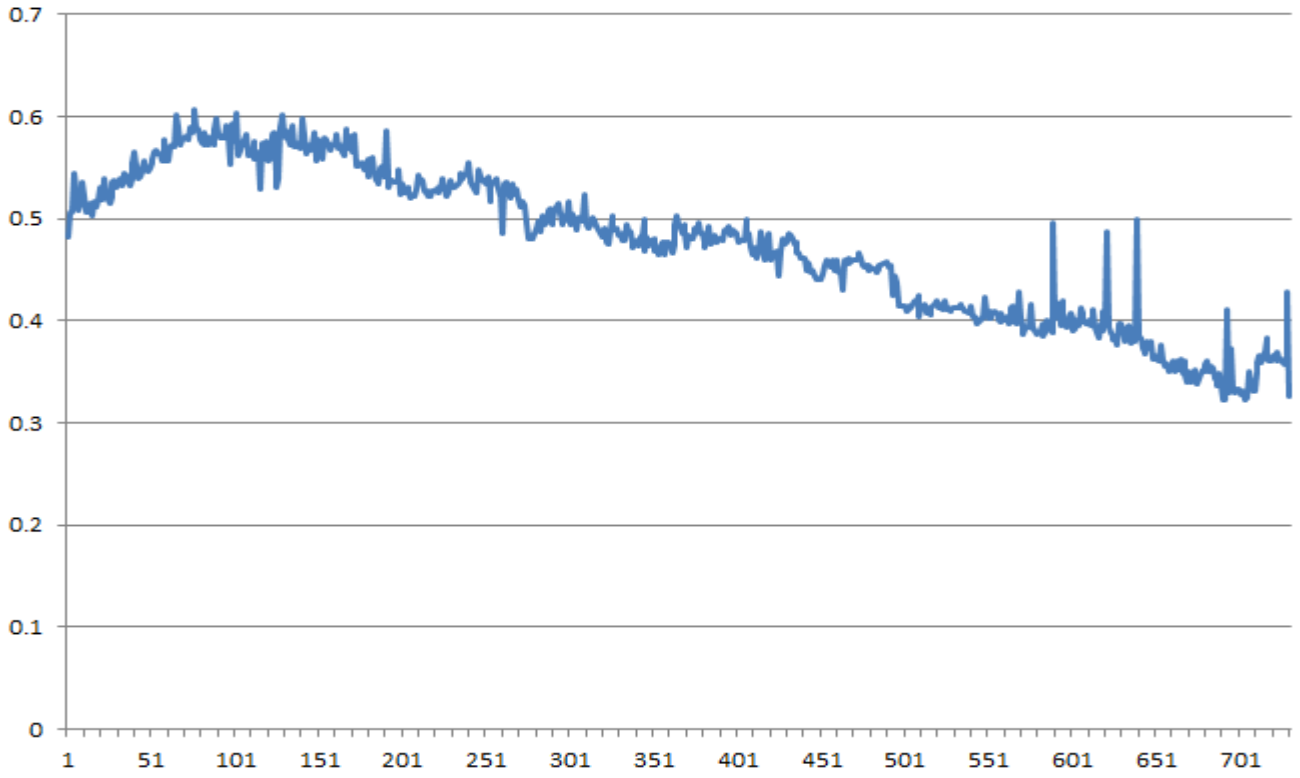
融合伪代码1

```
val conf = new SparkConf().setMaster(……).setAppName(……)
val ssc = new StreamingContext(conf, Seconds(60))


var stockEdgeRDD = getStockEdgeRDD()
var stockCommRDD = getStockCommRDD()

val onlineDataFlow = getDataFlow(ssc.sparkContext)
val edgeStreamRDD = ssc.queueStream(onlineDataFlow, true)
```

```
edgeStreamRDD.foreachRDD {
  rdd => {
    val incEdgeRDD  = buildIncEdgeRDD(rdd)

    val newEdgeRDD  = stockEdgeRDD.union(incEdgeRDD)
    val newCommRDD  = newEdgeRDD.join(stockCommRDD)
    val neighborRDD = countNeighbor(newCommRDD)


    val newCommRDD  = neighborRDD.map(
            case(vid, neighbors) => (vid, vote(neighbors))
    )
    stockCommRDD = newCommRDD
    stockEdgeRDD = newEdgeRDD
```

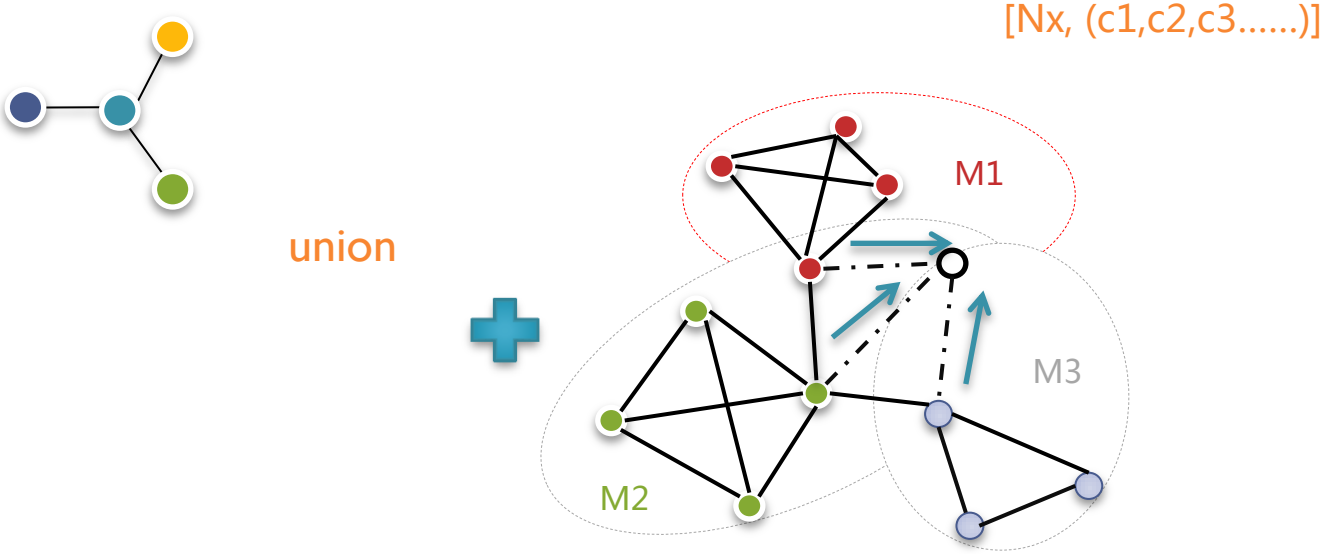
```
ssc.start()
ssc.awaitTermination()
```

融合效果1（Join & Vote）



斯坦福数据集：http://snap.stanford.edu/data/as.html

融合模型2（BMG）



- Broadcast & Modularity Greedy

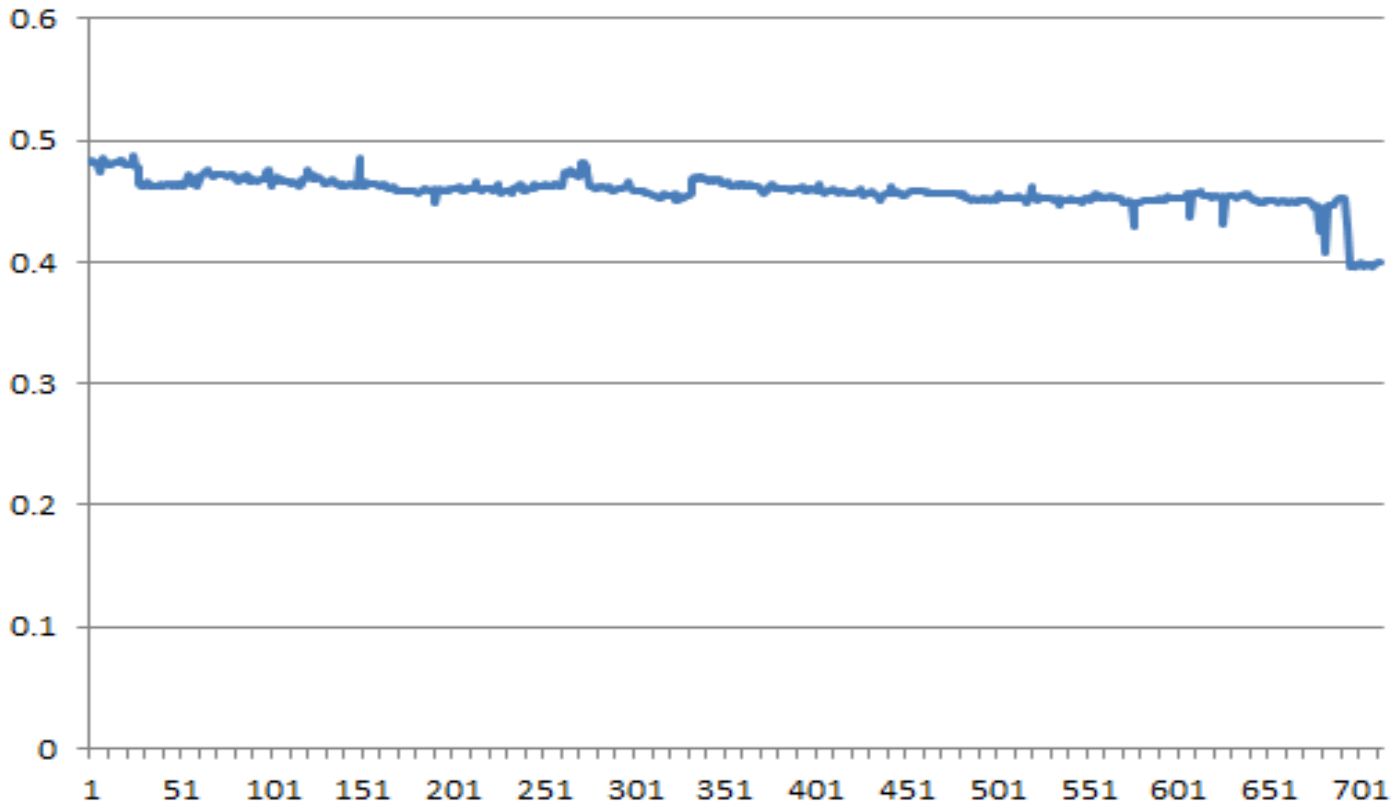
融合代码2（BMG)

```
val conf = new SparkConf().setMaster(……).setAppName(……)
val ssc = new StreamingContext(conf, Seconds(60))


val onlineDataFlow = getDataFlow(ssc.sparkContext)
val edgeStreamRDD  = ssc.queueStream(onlineDataFlow, true)

var stockCommRDD = getStockCommRDD()
var stockGraph = initGraph(getStockEdgeRDD(), stockCommRDD)
```

```
edgeStreamRDD.foreachRDD {
  rdd => {
   val incGraph  = buildIncGraph(rdd)
   var newGraph  = unionGraph(incGraph, stockGraph)
   val incCommRDD = newGraph.mapReduceTriplets[Array[VertexData]](commPropFunc, _ ++ _)
   .join(incGraph.vertices)
   .map {
       case (vid, (vdArray,vd)) => (vid, getBestCommunity(vdArray, curGraphTotalDegree))
   }
   stockGraph = updateCommunityAndRefreshGraph(incCommRDD, newGraph, vertexWeightRDD)
   stockCommRDD = merge(stockCommRDD, incCommRDD)
   rdd
  }
}
```

```
ssc.start()
ssc.awaitTermination()
```

融合效果2（BMG）



斯坦福数据集：http://snap.stanford.edu/data/as.html

融合性能1（Join & Vote）

- 每分钟增量平均边数：10w级别

- Worker: 100
- Driver-Memory: 20G
- Core: 2
- Executor-Memory: 20G

处理周期：2秒~

融合准确度1(Join & Vote)

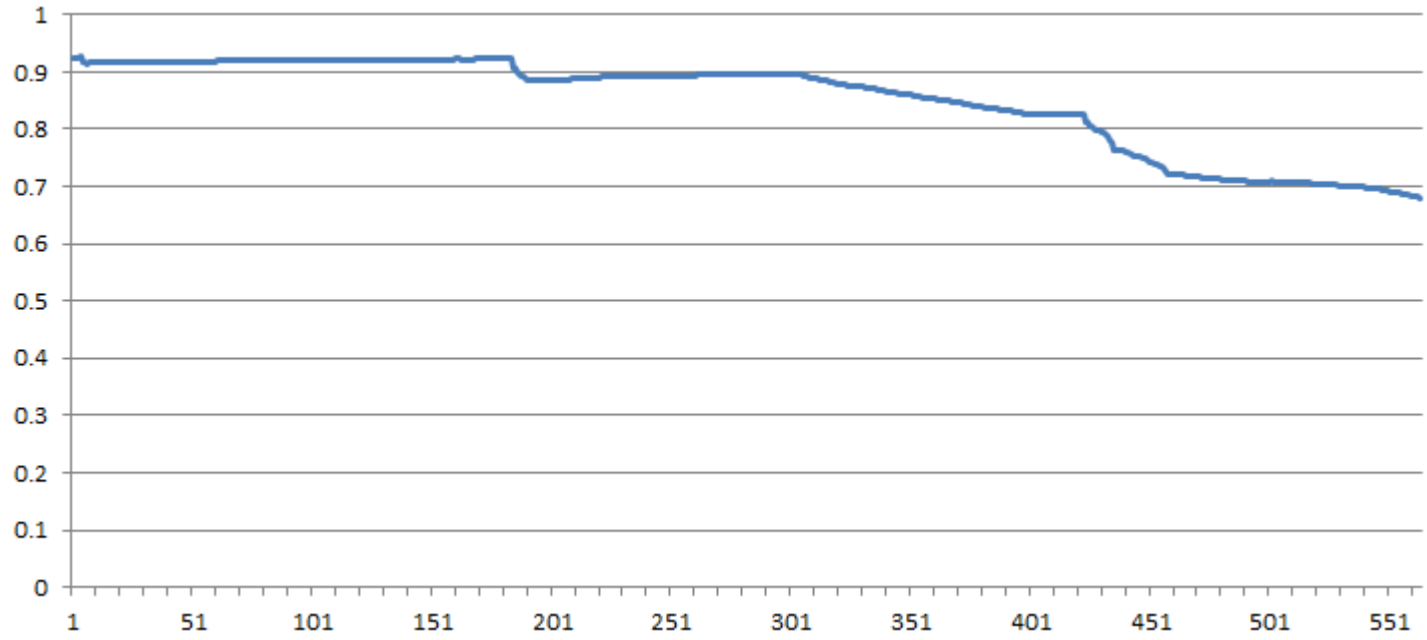
融合性能2（BMG）

- 每分钟增量平均边数：10w级别

  - Worker: 100
  - Driver-Memory: 20G
  - Core: 2
  - Executor-Memory: 20G

处理周期：40秒~

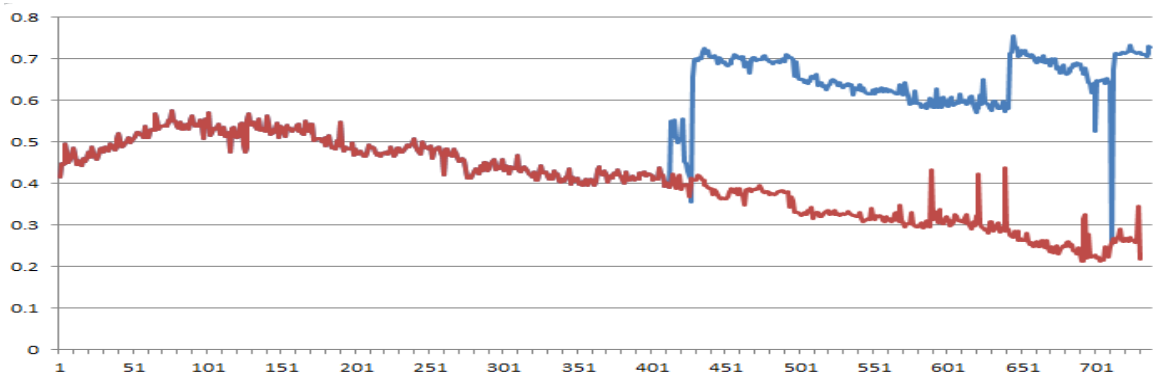
融合准确度2（BMG）


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
51
101
151
201
251
301
351
401
451
501
551

实时效果修正

- **监控修正**
  - 对Modularity做监控，当整体值降低到阈值以下，做一次全量的离线社区划分，并进行数据补全

- **定时修正**
  - 每天进行一次离线全量的社区划分，直接将结果切换为新一天的实时任务的基础数据

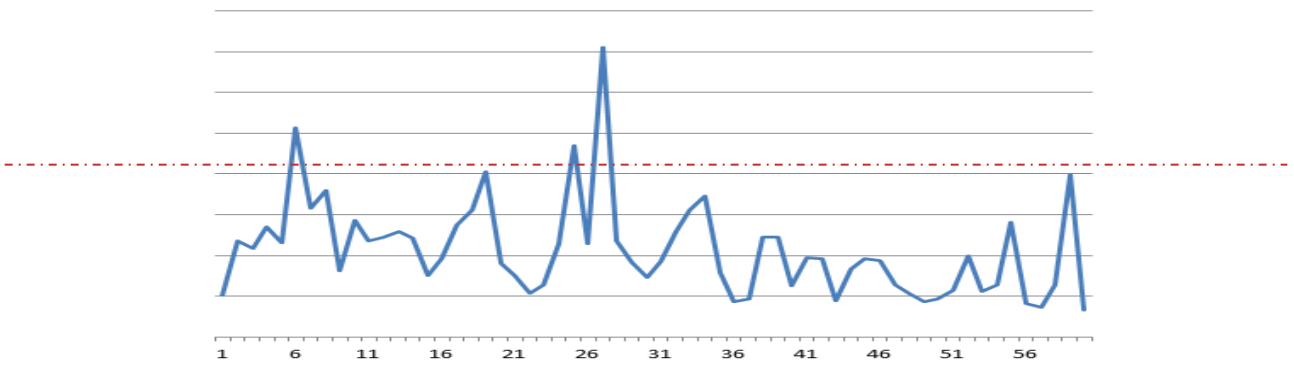


流图合璧——优点

- **好处**
  - **接口一体化**
    - 基于Spark栈的Streaming和GraphX接口，无缝结合
  - **模型细腻化**
    - 使用强大的图接口，在流中进行图操作和模型构建，获得更好的准确度和效果
  - **性能优化**
    - 利用图算子，进行图结构优化，可以避免进行RDD的耗时操作

流图合璧——注意点

- **注意点**
  - **波动和尖刺**
    - 线上真实环境中，每周期的数据量会有波动的现象
    - 根据1周内的每天每个周期运行时输入数据量和处理时间，计算出系统最佳负荷
  - **资源保障**
    - 合理配置Streaming任务的Worker、Core和Memory，保证大多数情况不会出现严重的延迟



流图合璧——注意点

- **注意点**
  - **假死**
    - 在流作业中，对图进行大规模消息传递，可能会导致流作业假死，需要谨慎限制消息规模
  - **数据堆积**
    - 当一个周期的输入数据，超出系统处理能力，数据会产生堆积，需要顺延接下一个周期的数据处理
    - 目前的处理方法是数据缓冲池

Q & A