第十五讲 图处理系统Giraph

徐 辰
cxu@dase.ecnu.edu.cn
華東師範大學

图数据

2

- 数据本身以图的形式呈现
  - 社交网络
  - 传染病传播途径
  - 交通路网
- 某些非图结构的数据，也可以转换为图模型后进行处理
  - 网页链接（将网页视为顶点，链接视为边）
  - 机器学习训练数据（数据项看作顶点，顶点之间没有边）

关联性分析

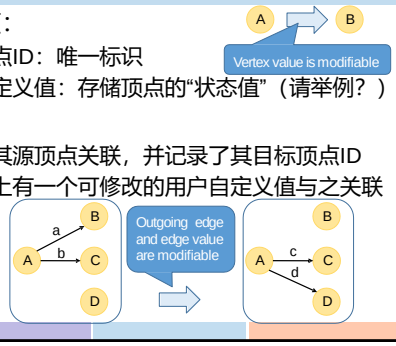
3

- 图数据结构表达了数据之间的关联性
- 通过获得数据的关联性，抽取有用的信息
  - 购物通过为购物者之间的关系建模，就能很快找到偏好相似的用户，并为之推荐商品
  - 图在社交网络中，通过传播关系发现意见领袖

图结构

4

- 顶点：
  - 顶点ID：唯一标识
  - 自定义值：存储顶点的“状态值”（请举例？）
- 边：
  - 和其源顶点关联，并记录了其目标顶点ID
  - 边上有一个可修改的用户自定义值与之关联



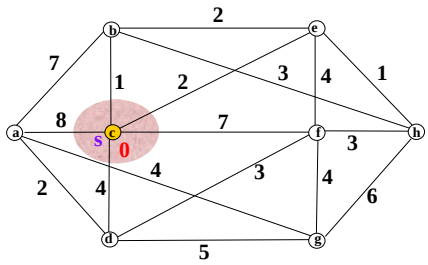
图算法

5

- 典型图算法：
  - 单源最短路径
  - 连通分量
  - 遍历…
- 求最短路径的Dijkstra算法
  - 输入：连通带权图G，$|V_G|=n$，指定顶点$s \in V_G$
  - 输出：每个顶点v的标注$(L(v), u)$，其中：
    - $L(v)$即从s到v的最短路径长度（目前可得的）
    - u是该路径上v前一个顶点。

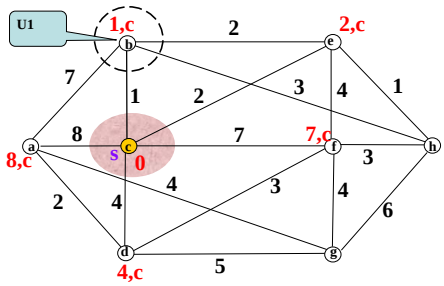
最短路径

6



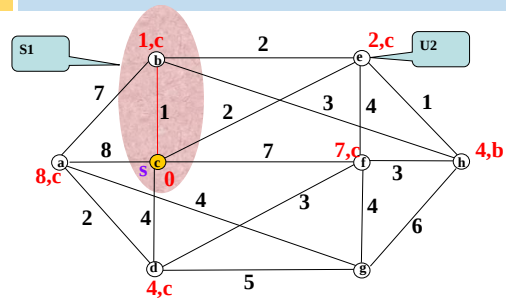
最短路径

7



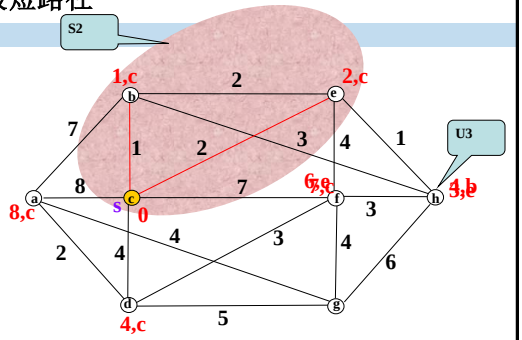
最短路径

8



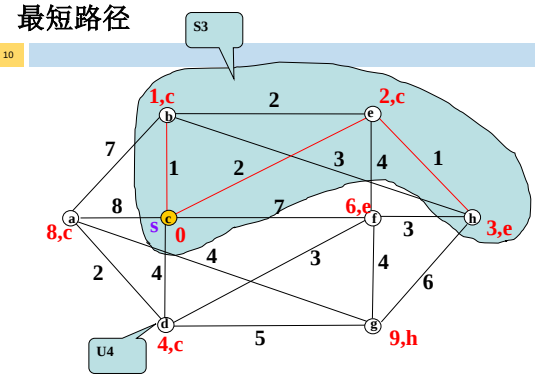
最短路径

9



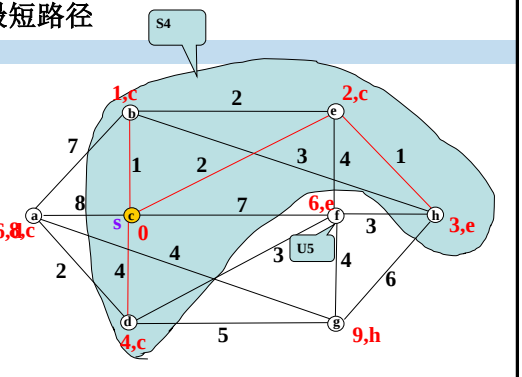
最短路径

10



最短路径

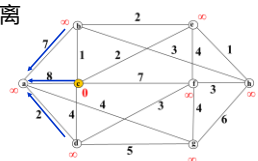
11



图算法共性

12

- 共性：顶点给邻居传递消息，不断进行更新，此过程迭代，直到最终收敛
  - 集中式串行算法：依次更新每个顶点与源点之间的距离
  - 分布式并行算法：同时更新所有顶点与源点之间的距离



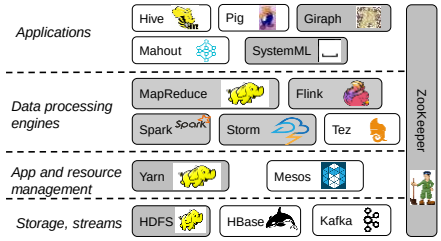
Pregel/Giraph简介

13

- 基于BSP模型实现的分布式图处理系统
  - 一套可扩展的、有容错机制的平台
  - 提供一套灵活的API，描述图计算，比如图遍历、最短路径、PageRank计算等
- 历程
  - 2010年，Google发表Pregel论文
  - 2012年，Giraph开源实现
    - 利用MapReduce框架
    - 不是基于MapReduce API计算

分布式计算系统生态圈

14



大纲

15

- 设计思想
- 体系架构
- 工作原理
- 容错机制

设计考虑-1

16

- 对图处理算法通用
  - 由于图处理系统需要具备一定的通用性，这就使得为特定的图应用定制相应的分布式框架的方式不可行。
  - 例如，我们专门设计一个针对PageRank计算的图处理系统，针对图遍历设计另一个专用的系统。显然，对于新的图算法或者图相关的应用，需要做大量的重复实现。

设计考虑-2

17

- 支持大规模图处理
  - 由于图处理系统需要处理大规模图数据，所以使用BGL等单机的图算法库无法满足要求。
  - 这是因为单机的存储容量有限，单机的图算法库对所能处理的图的规模有很大的限制。

设计考虑-3

18

- 自身具备容错能力
  - 由于图处理系统需要具备容错能力，所以使用并行BGL等图计算库也无法满足要求。
  - 类似于MPI，这些并行图计算库仅提供了并行图处理的方式，并不能够有效支持容错。

设计考虑-4

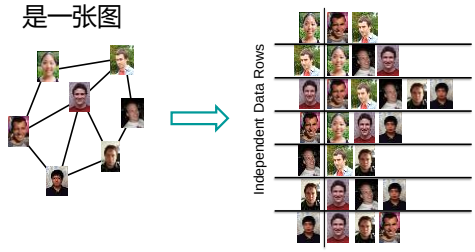
19

- 为图处理进行优化
  - 我们使用MapReduce来编写程序进行图处理能够满足前述几项要求，但是MapReduce作为一个通用的分布式处理系统，并未就图处理进行有针对性的优化。
    - 从用户编程角度来看，MapReduce将图数据看作是普通的键值对，图遍历（BFS）、最短路径（SSSP）、PageRank等算法都需要使用MapReduce函数实现，逻辑复杂，易用性差。
    - 从系统性能角度来看，利用MapReduce的实现迭代式图算法往往需要反复读写HDFS，效率低下。

MapReduce中图的表示

20

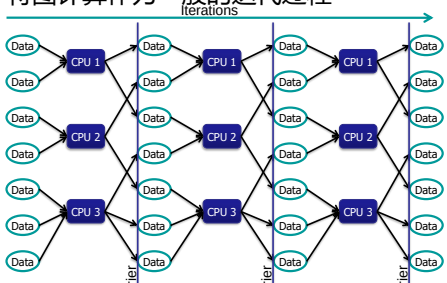
- 用户编写代码将图转变为key-value格式的文件，MapReduce系统并不认为这个文件是一张图



MapReduce图处理

21

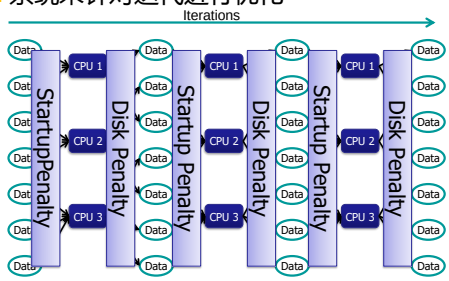
- 将图计算作为一般的迭代过程



MapReduce图处理

22

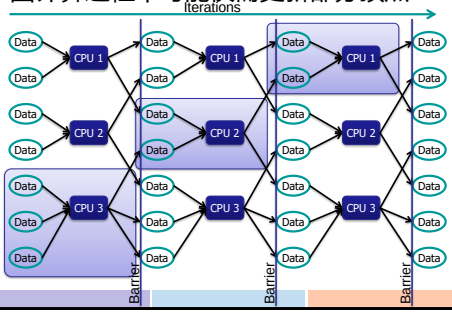
- 系统未针对迭代进行优化



MapReduce图处理

23

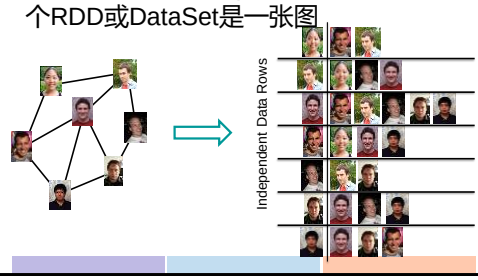
- 图计算过程中可能仅需更新部分顶点



Spark/Flink中图的表示

24

- 编写代码将图转变为由key-value对组成的RDD或DataSet，Spark/Flink并不认为这个RDD或DataSet是一张图



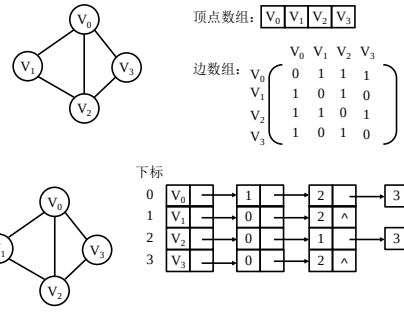
大纲

25

- 设计思想
  - 数据模型
  - 计算模型
  - 迭代模型
- 体系架构
- 工作原理
- 容错机制

邻接表与邻接矩阵

26



Giraph图数据模型

27

- vertexID：顶点具有唯一的标识
- Value：与顶点关联一个自定义的计算值，用于存储计算过程中所需保持的数值
  - PageRank算法中顶点的Rank值
  - SSSP算法中顶点与起始点间的距离等
- Weight：图中的边由其源顶点和目标顶点确定，边上也有用户自定义的权值

Giraph图数据模型

28

- 图中的某一顶点及以其为源点的边可以抽象地表示如下：

  [vertexID, value, [targetID$_1$,weight$_1$], [targetID$_2$,weight2]…]

- [vetextID, targetID]表示一条边

大纲

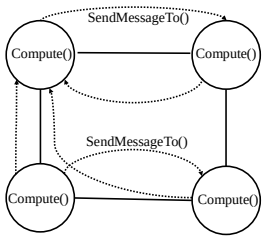
29

- 设计思想
  - 数据模型
  - 计算模型
  - 迭代模型
- 体系架构
- 工作原理
- 容错机制

计算模型：Vertex-centric

30

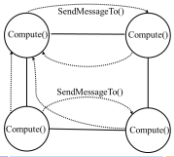
- Giraph以顶点为主进行数据表示，采用以顶点为中心（Vertex-centric）的计算模型



计算模型：Vertex-centric

31

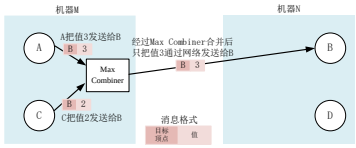
- 图中的边并不是核心对象，在边上面不会运行计算，只有顶点才会执行用户自定义函数进行相应计算
  - Compute() 用户定义的计算函数
  - SendMessageTo() 消息传递给指定顶点



Combiner:可选

32

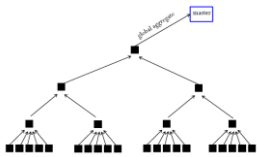
- Combiner根据定义的combine()函数将发往同一顶点的多个消息合并成一个消息，减少了传输和缓存的开销



Aggregator:可选

33

- 一种全局通信、监控和数据查看的机制
  - 每个顶点都可以向某个Aggregator提供数据，系统可以对这些值进行聚合操作产生一个值，之后所有顶点都可以获取这个值



大纲

34

- 设计思想
  - 数据模型
  - 计算模型
  - 迭代模型
- 体系架构
- 工作原理
- 容错机制

顶点计算值的更新

35

- 在单机编写图算法时，我们可以使用循环的形式串行地更新图中顶点的计算值。
- 分布式场景下图的顶点分布在不同的物理节点或在同一物理节点由不同处理器更新计算值，因而更新计算值的过程是并行的

图算法的迭代

36

- 图的算法都是由若干轮迭代组成的过程
- 某一顶点的计算值在本轮迭代更新后而可能其它顶点的计算值尚未完成该轮计算，那么该顶点是否可以立即进入下一轮迭代计算?
  - 不可以，BSP模型（Bulk Synchronous Parallel）
  - 可以，非BSP模型（本课程不讨论）

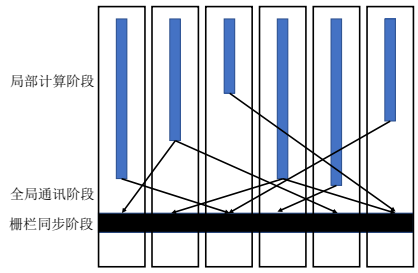
Bulk Synchronous Parallel(BSP)模型

37

- BSP模型设计准则是整体同步，引入了超步(superstep)概念
- 从整体来看,一个BSP程序由一系列串行的超步组成，一个超步分为三个阶段：
  - 局部计算阶段，每个处理器只对存储本地内存中的数据进行本地计算
  - 全局通信阶段，对任何非本地数据进行操作
  - 栅栏同步(Barrier Synchronization)阶段，等待所有通信行为的结束。

Bulk Synchronous Parallel(BSP)模型

38



Pregel BSP

39

- Pregel/Giraph的计算过程由一系列被称为超步(superstep)的迭代(iterations)组成
- 对于每一个超步,
  - 在局部计算阶段，各个处理器对每个顶点调用用户自定义的函数Compute()，该函数可以读取超步(S-1)中发送给某一顶点V的消息，并且描述顶点V在一个超步S中需要执行的操作。在此过程中修改顶点的计算值value及其出射边的权值weight

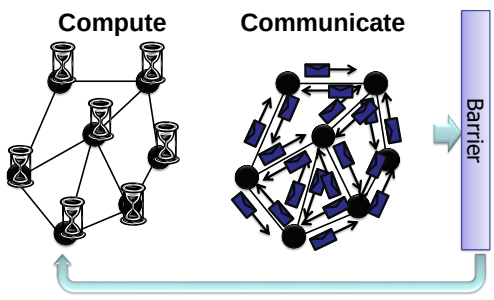
Pregel BSP

40

- 在全局通信阶段，针对每个顶点调用的SendMessageTo()函数包含了其需要传送给其它顶点的内容，各个处理器之间通信传输这些消息内容，从而将消息给其它顶点，其它顶点获取这些消息将用于在下一个超步(S+1)的计算
- 在栅栏同步阶段，每个顶点都要等待其它所有顶点完成计算并发出消息。

Pregel BSP

41



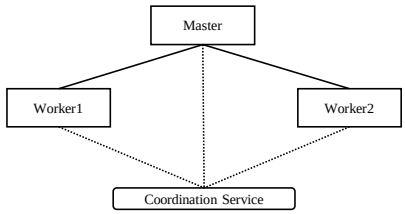
大纲

42

- 设计思想
- 体系架构
  - 架构图
  - 应用程序执行流程
- 工作原理
- 容错机制

Pregel抽象架构图

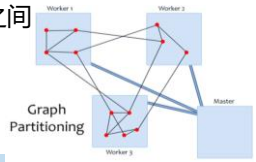
43



系统角色

44

- Master：负责管理各个Worker执行任务
- Worker：系统将图进行了划分，形成若干个分区，每个Worker负责一个或多个分区并负责针对该分区的计算任务
- Coordination Service：协调Master与worker以及worker之间



Worker

45

- 在一个Worker中，它所管辖的分区的顶点描述信息是保存在内存中的
  - 顶点与边的值：包括顶点的当前计算值，以该顶点为起点的出射边列表，每条出射边包含了目标顶点ID和边的值
  - 消息：包含了所有接收到的、发送给该顶点的消息
  - 标志位：用来标记顶点是否处于活跃状态。如果一个顶点V在超步S接收到消息，那么V将参与下一个超步的计算，意味着在下一个超步S+1中（而不是当前超步S中）处于“活跃”状态

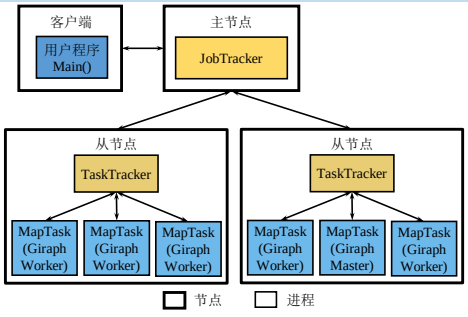
Master

46

- Master负责协调各Worker执行任务，每个Worker均向Master发送自己的注册信息，Master会为Worker分配一个唯一的ID
- Master维护所有Worker的各种信息，包括每个Worker的ID和地址信息，以及每个Worker被分配到的分区信息
- Master维护的数据信息的大小，只与分区的数量有关，而与顶点和边的数量无关

Giraph与MapReduce

47



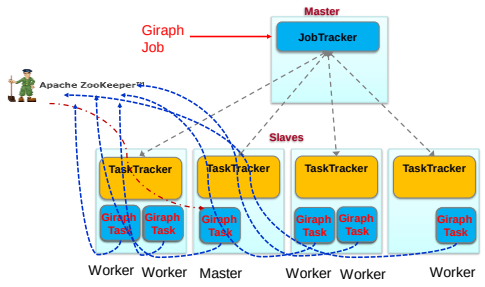
Giraph与MapReduce

48

- Giraph参考了Pregel体系架构，但在实现时借用了MapReduce框架启动Master和Worker这些进程
- Giraph并没有像普通MapReduce程序那样编写Map和Reduce 两个API函数，而是将所有的图处理逻辑都在启动Map任务的run函数中实现。从MapReduce框架的角度来看，执行Giraph作业仅启动了Map任务

ZooKeeper in Giraph

49



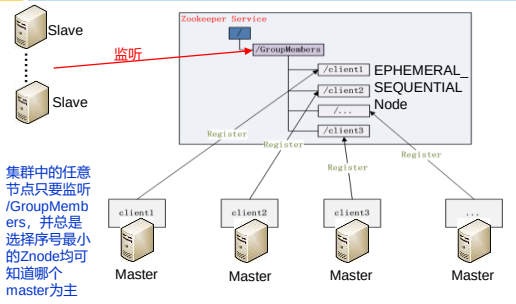
ZooKeeper in Giraph

50

- 分布式选主：启动的Map任务中，有一个作为Giraph的Master，其余作为Worker。如何决定哪个Map任务作为Master?
- 栅栏同步：Zookeeper还可以实现BSP模型中栅栏同步的功能

回顾：多master选主

51



集群中的任意节点只要监听/GroupMembers，并总是选择序号最小的Znode均可知道哪个master为主

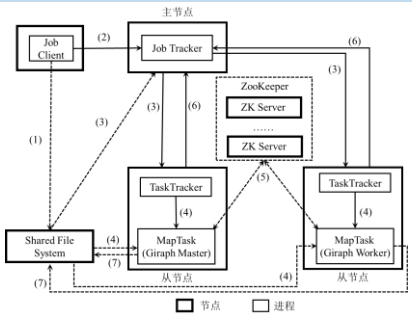
大纲

52

- 设计思想
- 体系架构
  - 架构图
  - 应用程序执行流程
- 工作原理
- 容错机制

应用程序执行流程

53



应用程序执行流程

54

1. Client将用户编写的Giraph作业配置信息、jar包等上传到共享的文件系统(HDFS)
2. 通过Client提交给JobTracker，告诉JobTracker作业信息的位置
3. JobTracker读取作业的信息，生成一系列Map任务，调度给有空闲slot的TaskTracker
4. TaskTracker根据JobTacker的指令启动Child进程执行Map任务，Map任务将从共享文件系统读取输入数据

应用程序执行流程

55

5. 多个Map任务通过ZooKeeper进行选主，其中一个Map任务作为Giraph的Master，其它作为Giraph的Worker，并且Master和Worker通过Zookeeper协调并执行Giraph作业
6. JobTracker从TaskTracker处获得Giraph的Master和Worker进度信息
7. 任务完成计算后将结果写入共享文件系统，则意味着整个作业执行完毕

大纲

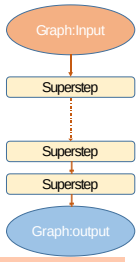
56

- 设计思想
- 体系架构
- 工作原理
- 容错机制

Giraph工作流程

57

- Giraph读入数据经过一系列超步计算后将结果输出，可以大体划分为数据输入、迭代计算和数据输出三个阶段
  - 数据输入阶段：如何对输入的图数据进行划分？
  - 迭代计算阶段：如何进行超步计算以及同步控制？
  - 数据输出阶段：相对简单，各Worker对自己负责分区的计算结果写入HDFS等持久化存储系统



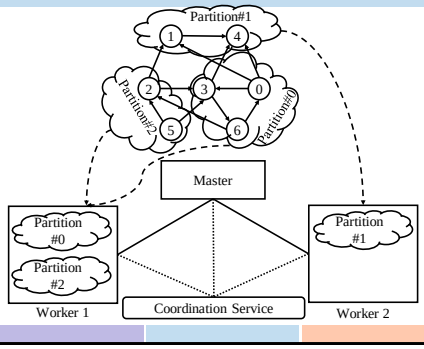
大纲

58

- 设计思想
- 体系架构
- 工作原理
  - 数据划分
  - 超步计算
  - 同步控制
- 容错机制

图数据的划分

59



图数据的划分

60

- Giraph需要将一张图根据顶点分解成多个分区，即按照顶点的ID决定该顶点属于哪一个分区，并且每个分区包含了一系列顶点和以这些顶点为起点的边
  - 系统默认：hash(ID) mod N
    - N为所有分区总数，
    - ID是这个顶点的标识符
- 无论哪一个Worker，只要将顶点的ID通过分区函数计算就可以知道该顶点属于哪个分区

输入数据分区与Giraph期望分区

61

- 输入数据的分区与Giraph需要的分区往往是不一致的
  - 例如存储在HDFS上的图数据以一个大文件的形成存在并按照顶点的ID排序，而Giraph系统希望顶点按照ID模取5进行划分。
- 因而，数据划分实际上是要完成输入数据到Giraph期望分区的调整，这一过程是由Master和Worker共同完成的

边加载边划分

62

- Master
  - 将输入图数据根据Worker数量初始分解为多个部分，例如以HDFS split为单位分解给
  - 分解后每个部分都是一系列记录（顶点和边）的集合，Master会为每个Worker分配部分数据
- Worker
  - 读取Master初始分配的部分图数据，并根据分区函数计算该顶点是否属于Worker负责的分区
  - 若不属于当前Worker负责的分区，那么当前Worker将该顶点发给其所属的分区所在Worker

大纲

63

- 设计思想
- 体系架构
- 工作原理
  - 数据划分
  - 超步计算
  - 同步控制
- 容错机制

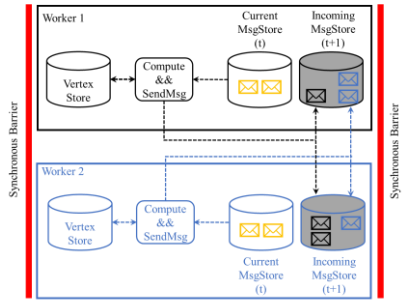
超步计算

64

- Worker为管辖的每个分区分配一个线程
- 对于分区中的每个“活跃”顶点，Worker根据前一个超步的消息，调用Compute()函数更新该顶点的计算值
- Compute()函数的计算过程中可以将更新后的计算值以消息形式发送给邻居顶点，但根据BSP模型这些消息仅会在下一个超步中使用

超步t

65



超步t中的MsgStore

66

- 在超步t中，Giraph需要存储两份消息
  - *MsgStore t* 存储在超步*t-1*全局通信阶段获取的消息，这些消息用于超步t中顶点的compute()计算
  - *MsgStore t+1*存储在超步*t*全局通信阶段获取的消息，意味着将用于超步*t+1*中顶点的compute()计算

超步t中的StatStore

67

- 在超步t中，Giraph也需要存储两份标志位信息
  - *StatStore t* 指示了当前超步*t*中处于活跃状态的顶点
  - *StatStore t+1*指示了超步*t+1*中处于活跃状态的顶点

超步t

68

- 超步t中某一顶点的处理过程：
  - 根据StatStore t中的某一活跃顶点，从MsgStore t读取属于该顶点的消息，并调用compute()函数更新VertexStore
  - 将更新后的计算值以消息形式发送给邻居顶点
  - 若无其它顶点发来的消息，则将StatStore t+1中该顶点的状态设置为非活跃顶点，意味着该顶点不参与下一个超步。否则将消息存入MsgStore t+1，用于下一个超步，并将StatStore t+1中该顶点的状态设置为活跃顶点

大纲

69

- 设计思想
- 体系架构
- 工作原理
  - 数据划分
  - 超步计算
  - 同步控制
- 容错机制

Worker间的同步

70

- Giraph中多个Worker同时进行超步计算，而多个Worker之间需要进行同步。
- 根据BSP模型，同步控制需要确保即所有Worker都完成超步t后再进入超步t+1。
- Giraph中的Master和Worker通过ZooKeeper来完成同步

回顾：Double Barrier双屏障

71

- 允许客户端在执行的开始和结束时同步。当足够的进程加入到双屏障时，进程开始计算。当执行结束时，离开屏障。
- 实现方法
  - 进入屏障：创建/ready和/process节点，每个进程都先到ready中注册，注册数量达到要求时，通知所有进程启动开始执行
  - 离开屏障：在/process下把注册ready的进程都建立节点，每个进程执行结束后删掉/process下对应节点。当/process为空时，执行结束

迭代的结束（收敛）

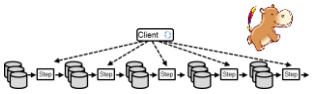
72

- 超步同步完成后，Worker根据StatStore t+1信息把在下一个超步还处于活跃状态的顶点数量报告给Master
- 如果Master收集到所有Worker中活跃顶点数量之和为0，意味着迭代过程可以结束。此时，Master会给所有Worker发送指令，通知每个Worker对自己的计算结果进行持久化存储。

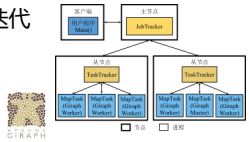
比较：Iterative MapReduce vs. Giraph

73

- Iterative MR：
  - Driver控制迭代
  - 多个job
  - 迭代中间结果反复读写HDFS



- Giraph：
  - Master(某个Task)控制迭代
  - 一个job
  - 迭代中间结果在内存



大纲

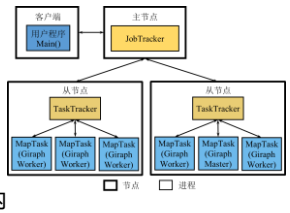
74

- 设计思想
- 体系架构
- 工作原理
- 容错机制

故障类型：MapReduce

75

- 主节点故障
  - JobTracker故障：如宕机引起
- 从节点故障
  - TaskTracker故障：如节点宕机引起
  - Task故障: 如JVM内存不够退出

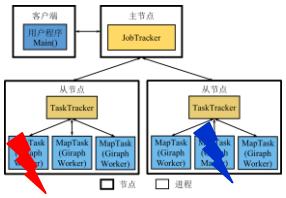


思考：是否可以直接利用MapReduce的容错机制?

故障类型：Giraph

76

- Master故障
  - 意味协调节点丢失
  - 怎么办?
- Worker故障
  - 意味计算节点丢失
  - 设置检查点



大纲

77

- 设计思想
- 体系架构
- 工作原理
- 容错机制
  - 检查点
  - 故障恢复
- 编程实例

检查点

78

- Giraph容许用户设置写检查点的间隔，即每隔多少超步写检查点
  - 默认情况下，该间隔的值为0，即不写检查点
  - 如果该间隔值不为0，那么每隔一定超步将进行写检查点。例如，如果检查点间隔设为4，那么在第0、4、8…超步将进行写检查点。
- 在需要写检查点的超步开始时，Master通知所有的Worker把管辖的分区状态（顶点、边、接收到的消息等），写入到持久化存储

大纲

79

- 设计思想
- 体系架构
- 工作原理
- 容错机制
  - 检查点
  - 故障恢复

故障恢复

80

- 在Giraph的实现中，当一个或多个Worker发生故障，被分配到这些Worker的分区信息就丢失了
- Giraph故障恢复方式（与Pregel论文中提出的confined recovery有差异）
  - Master将重新分配分区到其它存活的Worker，存活的Worker回滚到最近的检查点所在超步，假设为超步S
  - 在超步S开始时，从检查点中重新加载分区的信息并继续执行

问题

81

- Giraph中Worker蕴含在Map Task中，为什么不直接利用MapReduce的容错机制？
  - Map Task仅用于启动Giraph Worker，并非执行map()函数
  - 如果类似mapreduce那样在map()计算结束时写入磁盘，那么Giraph的计算也已经结束了
  - 如果每个superstep像map那样写磁盘，那么得到的是本地的备份
- 如果没有检查点，还能支持容错吗？

小故事：All roads lead to Rome

82

- 如果没有检查点，还能支持容错吗？

小故事：All roads lead to Rome

83

- 从柏林走到罗马，一路上随机出现野人攻击，如何走？

小故事：Pessimistic vs. Optimistic

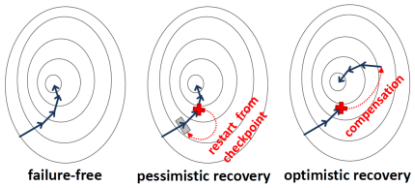
84

- Pessimus
  - 边走边建造城堡
  - 遇到野人攻击退回到城堡
  - 再从城堡出发重新走

- Optimus
  - 不建造城堡
  - 遇到野人攻击换条路走

小故事：Pessimistic vs. Optimistic

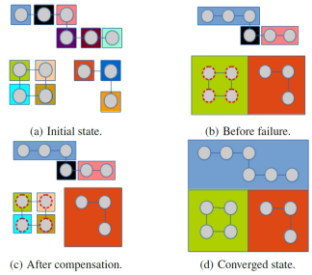
85



Sebastian Schelter et al. "All Roads Lead to Rome:" Optimistic Recovery for Distributed Iterative Data Processing. CIKM 2013

小故事：连通分量Optimistic Recovery

86



Sergey Dudoladov, Chen Xu, Sebastian Schelter, Asterios Katsifodimos, Stephan Ewen, Kostas Tzoumas, Volker Markl: *Optimistic Recovery for Iterative Dataflows in Action*. SIGMOD 2015.

课后阅读

87

- 论文
  - Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010). Pregel : A System for Large-Scale Graph Processing. In SIGMOD Conference (pp. 135–145).

本讲小结

88

- 设计思想
- 体系架构
- 工作原理
- 容错机制

谢谢！Q&A