大数据算法

第九讲 超越MapReduce的并行大数据处理

哈尔滨工业大学
王宏志
wangzh@hit.edu.cn

本讲内容

9.1 基于迭代处理平台的并行算法

9.2 基于图处理平台的并行算法

观察

- 观察: MapReduce已被证明作为一种非递归描述性语言的通用运行平台是成功的
  - HIVE (类SQL语言)
  - Pig (Rel algebra with nested types)

- 观察: 到处都是循环和迭代 :
  - 图, 聚类, 挖掘
- 问题:
  - MapReduce 无法表示循环和迭代
- 观察: 需要引入额外的循环:
  - 迭代由MapReduce代码外的脚本控制

基本想法

- 循环输出的结果很大
- 传递闭包
- PageRank (以收敛测试作为终止条件)

- 需要分布式不动点运算
  - 每次迭代需要一个额外的MapReduce过程

不动点

- 函数f不动点的功能是满足 $f(x) = x$的x

- 不动点查询 可以表达为关系代数加一个不动点运算
- Map - Reduce - 不动点
  - 猜想： 所有递归查询的模型

核心观点

- 想法:
  - 假设: 使MapReduce从循环角度达到最优
  - 设计递归语言的可扩展实现

- 更加精确的:
  - 通过最小化扩展，我们为递归语言提供有效地普通运行平台
  - ***Map, Reduce, 不动点***

三个迭代例子

例 1: PageRank(网络连接分析)

Rank Table R_0

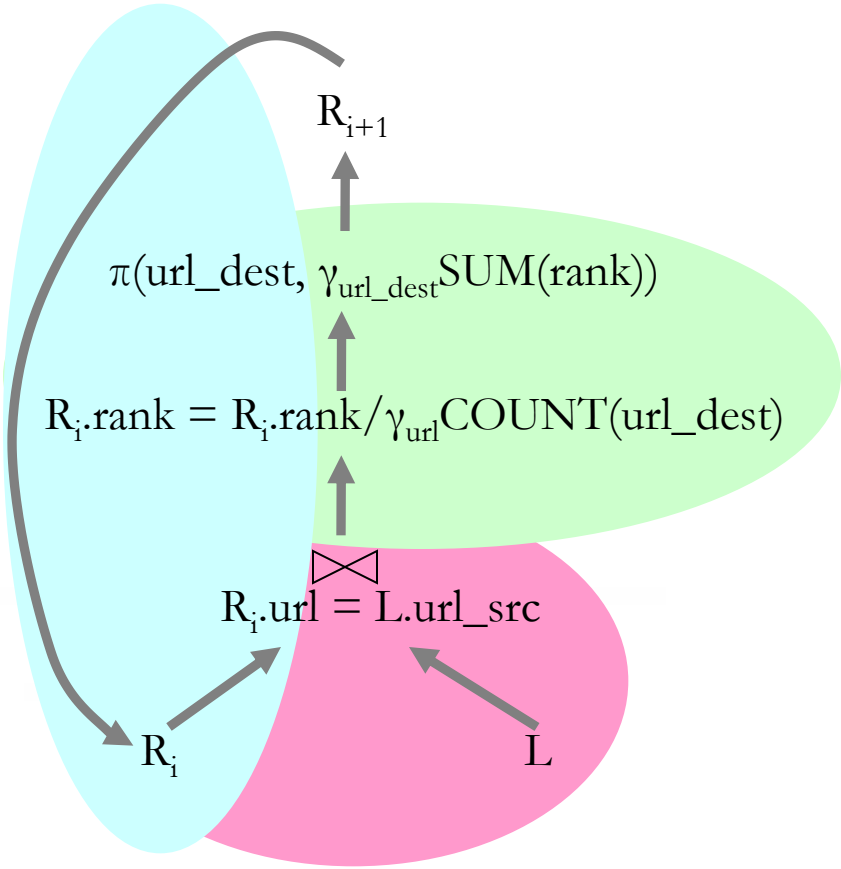
url	rank
www.a.com	1.0
www.b.com	1.0
www.c.com	1.0
www.d.com	1.0
www.e.com	1.0

Linkage Table *L*

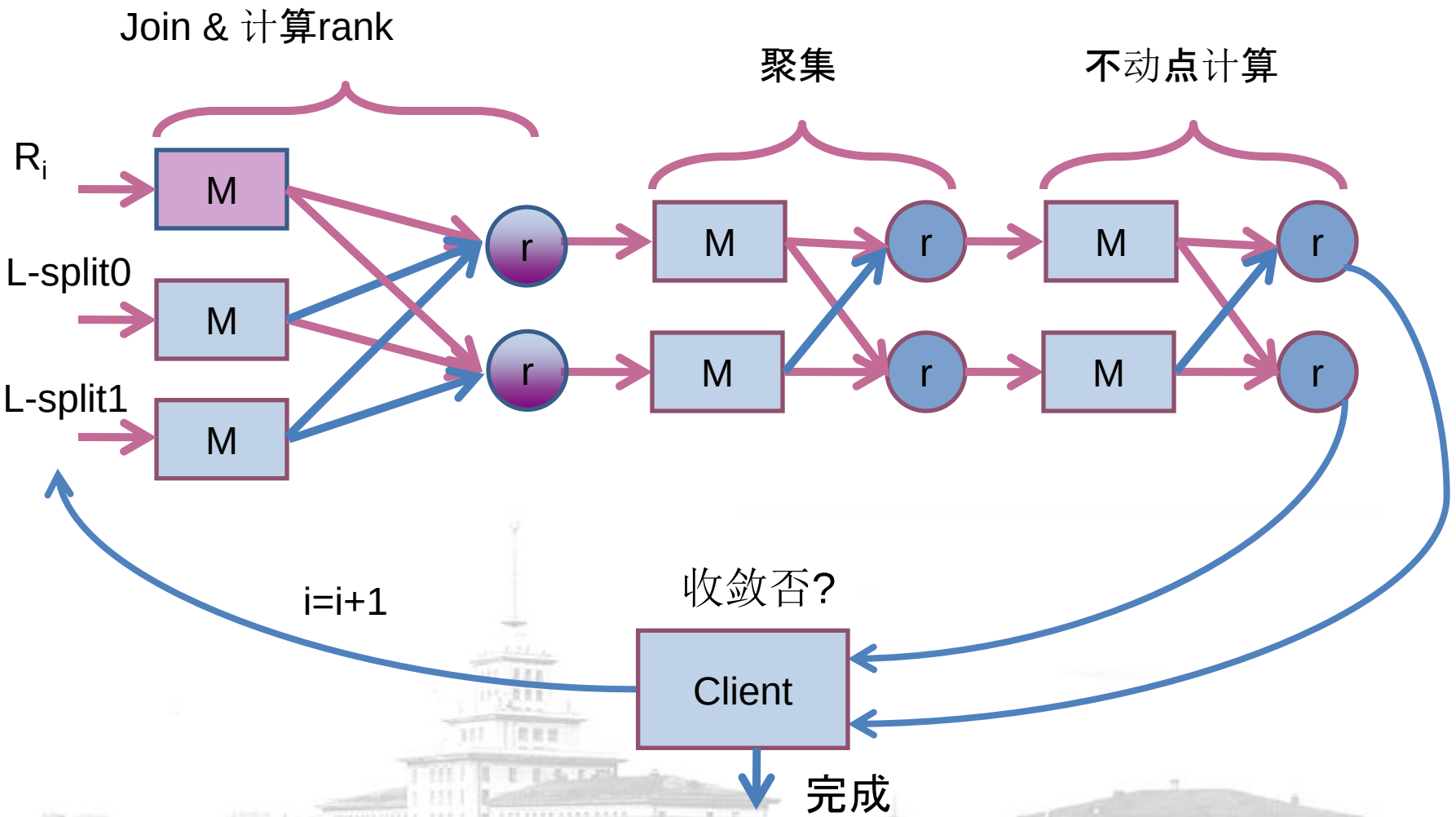
url_src	url_dest
www.a.com	www.b.com
www.a.com	www.c.com
www.c.com	www.a.com
www.e.com	www.c.com
www.d.com	www.b.com
www.c.com	www.e.com
www.e.com	www.c.com
www.a.com	www.d.com

Rank Table R_3

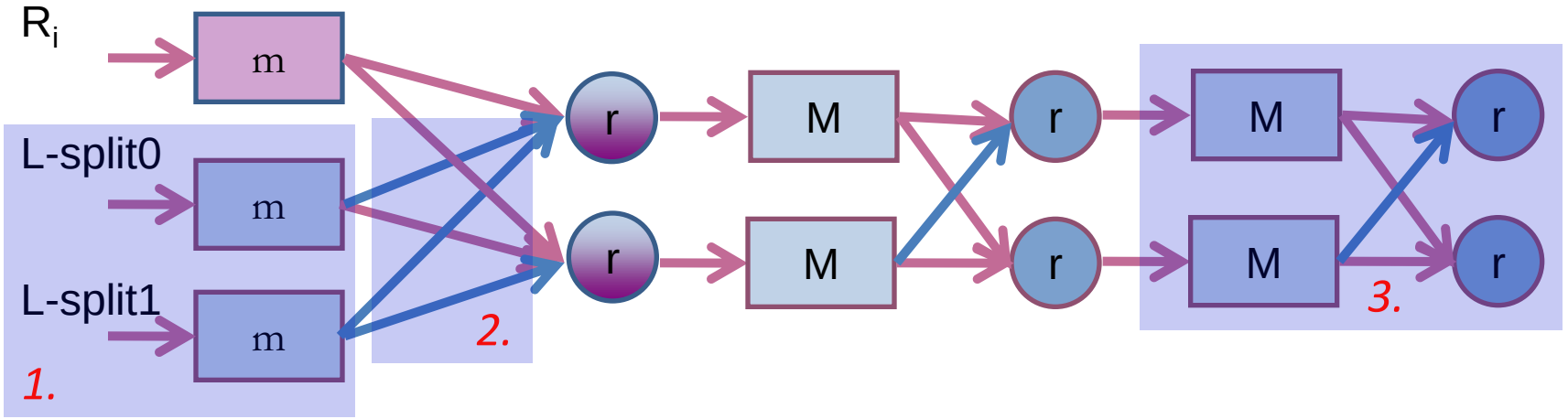
url	rank
www.a.com	2.13
www.b.com	3.89
www.c.com	2.60
www.d.com	2.60
www.e.com	2.13



MapReduce 的实现



出现了什么问题？



L (链表) 是循环不变的，但

1. 每轮载入L
2. 每次迭代重排L
3. 每次迭代中，不动点计算作为单独的MapReduce工作执行

例2: 传递闭包

Friend

name1	name2
Tom	Bob
Tom	Alice
Elisa	Tom
Elisa	Harry
Sherry	Todd
Eric	Elisa
Todd	John
Robin	Edward

找到所有Eric传递的朋友

R_0 {Eric, Eric}

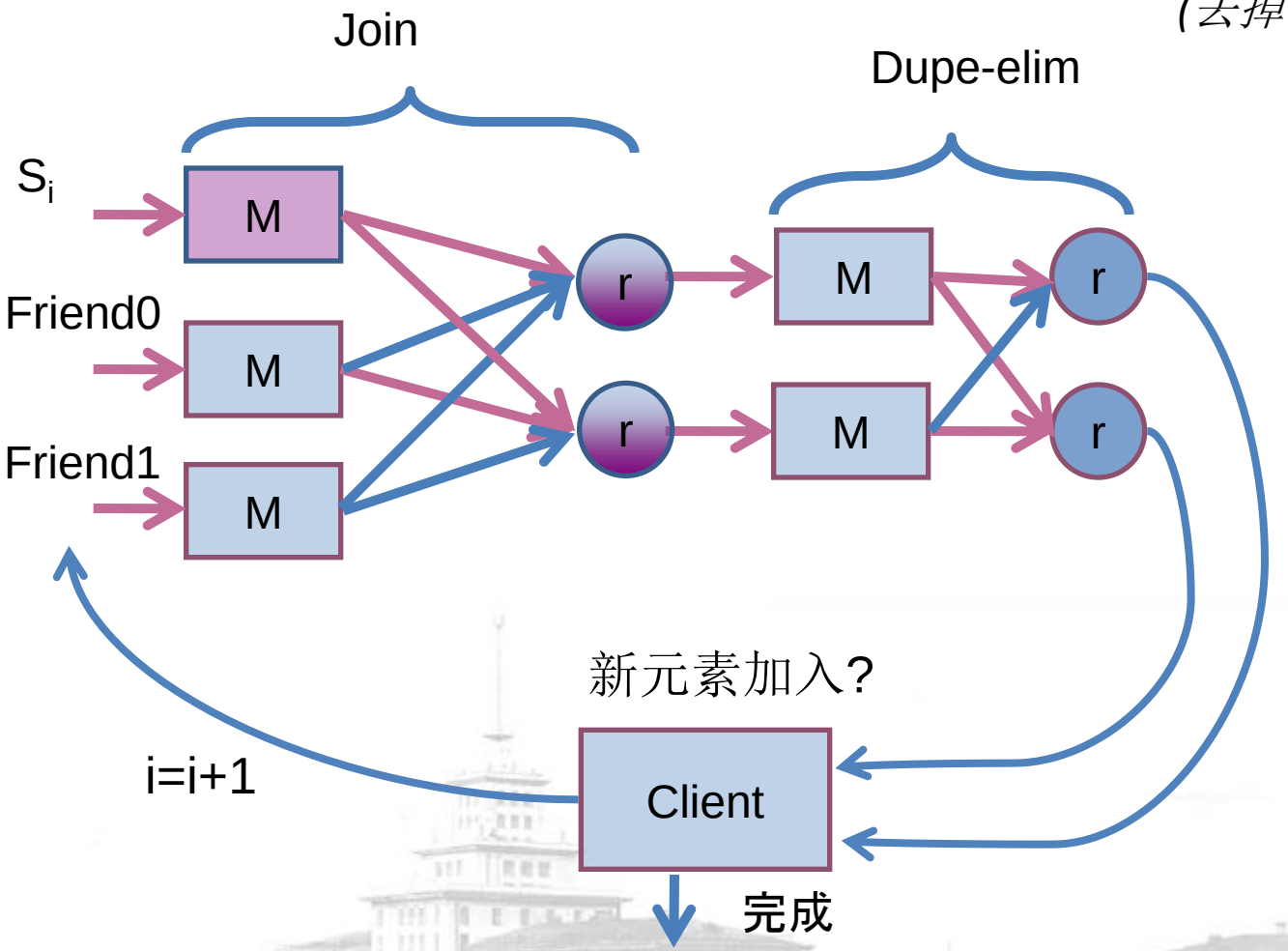
R_1 {Eric, Elisa}

R_2 {Eric, Tom
Eric, Harry}

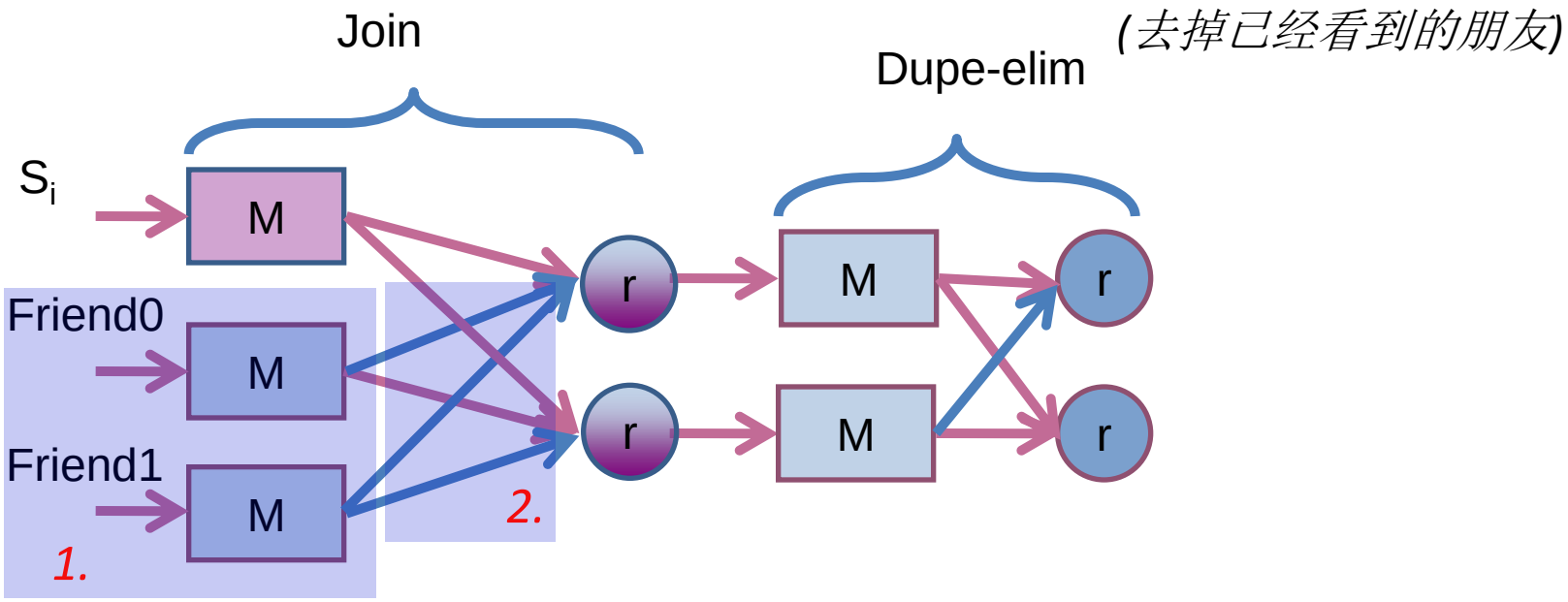
R_3 {}

例 2 MapReduce算法

(去掉已经看到的朋友)



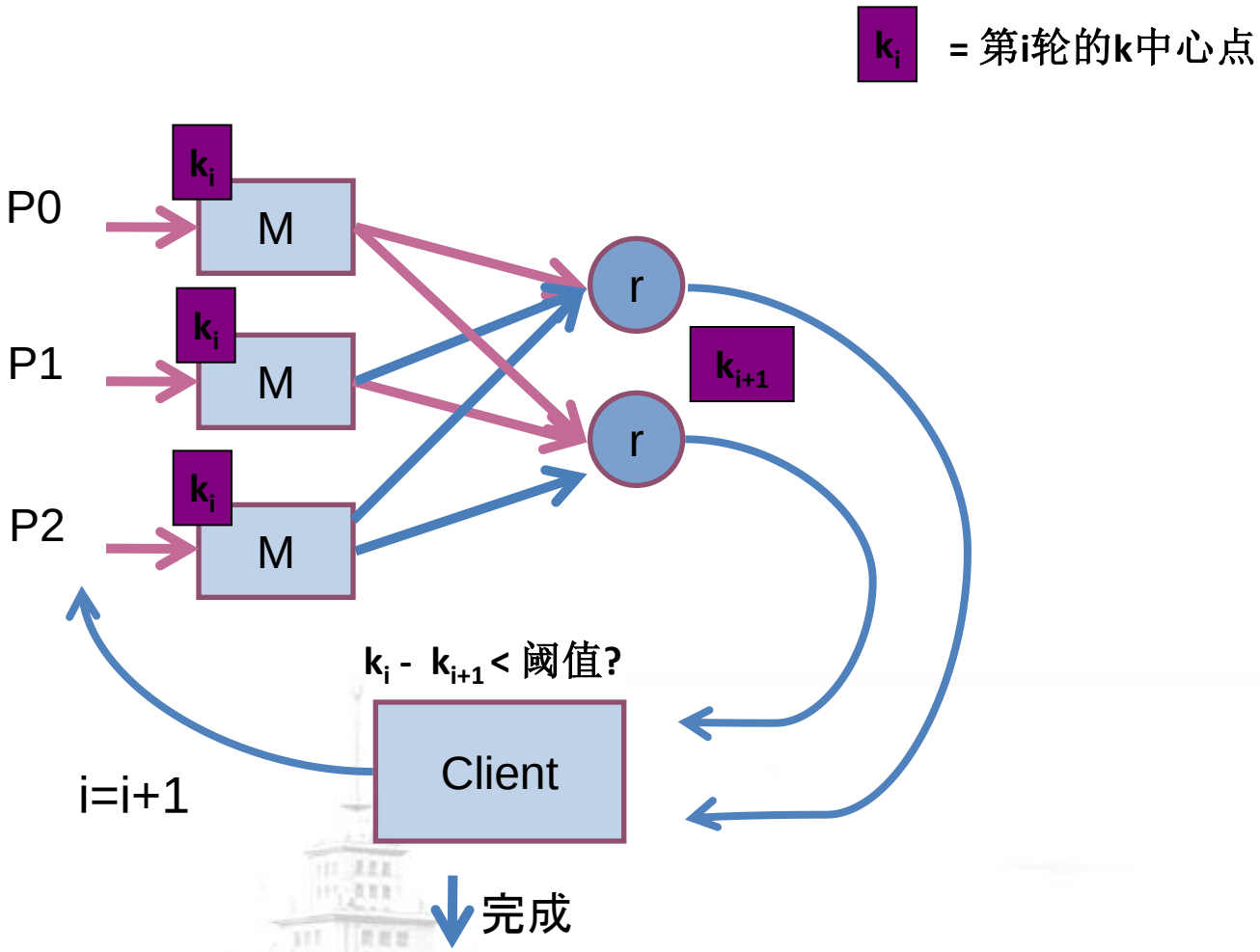
出现了什么问题



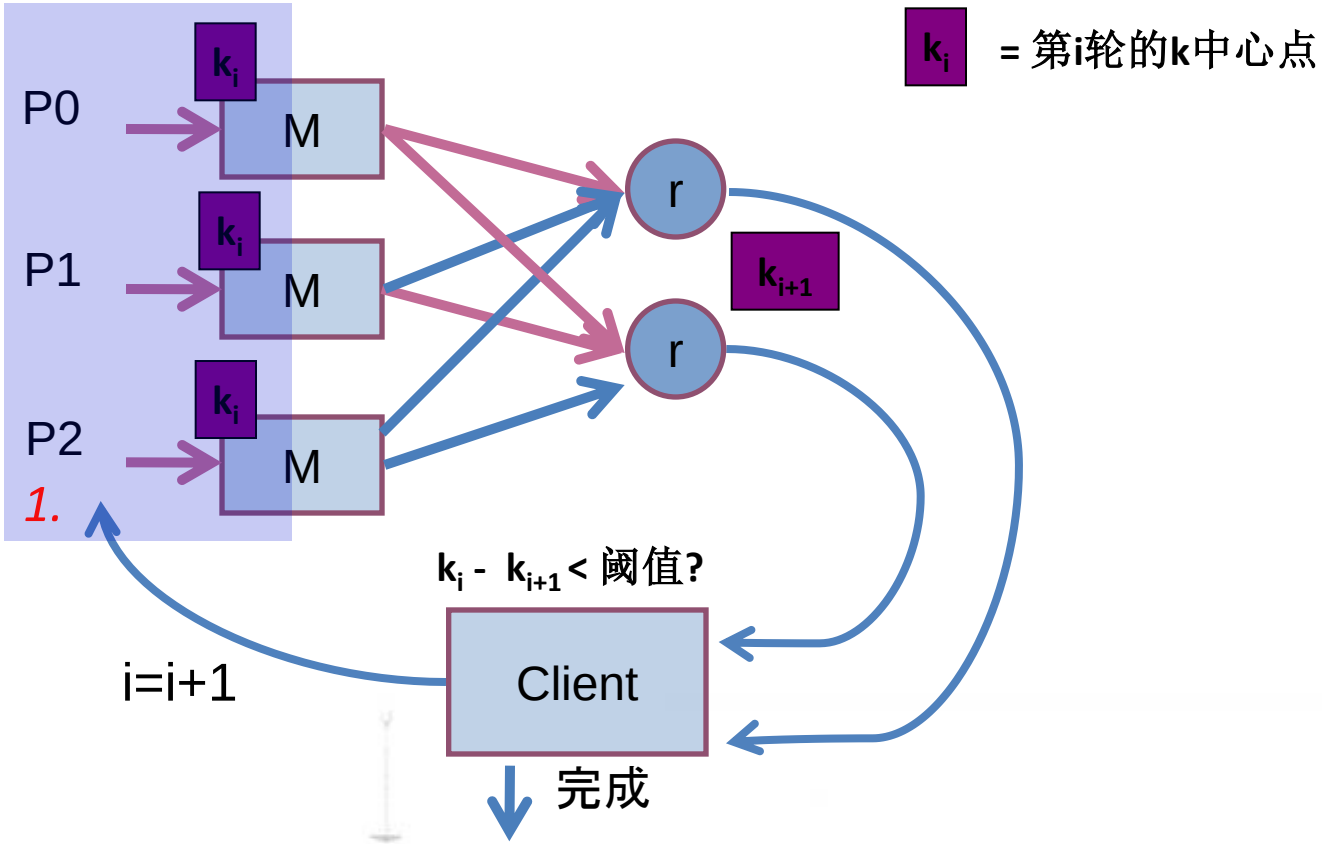
朋友是循环不变量

1. 每次循环载入朋友
2. 每次迭代重排朋友

例3: k-means算法



出现了什么问题？



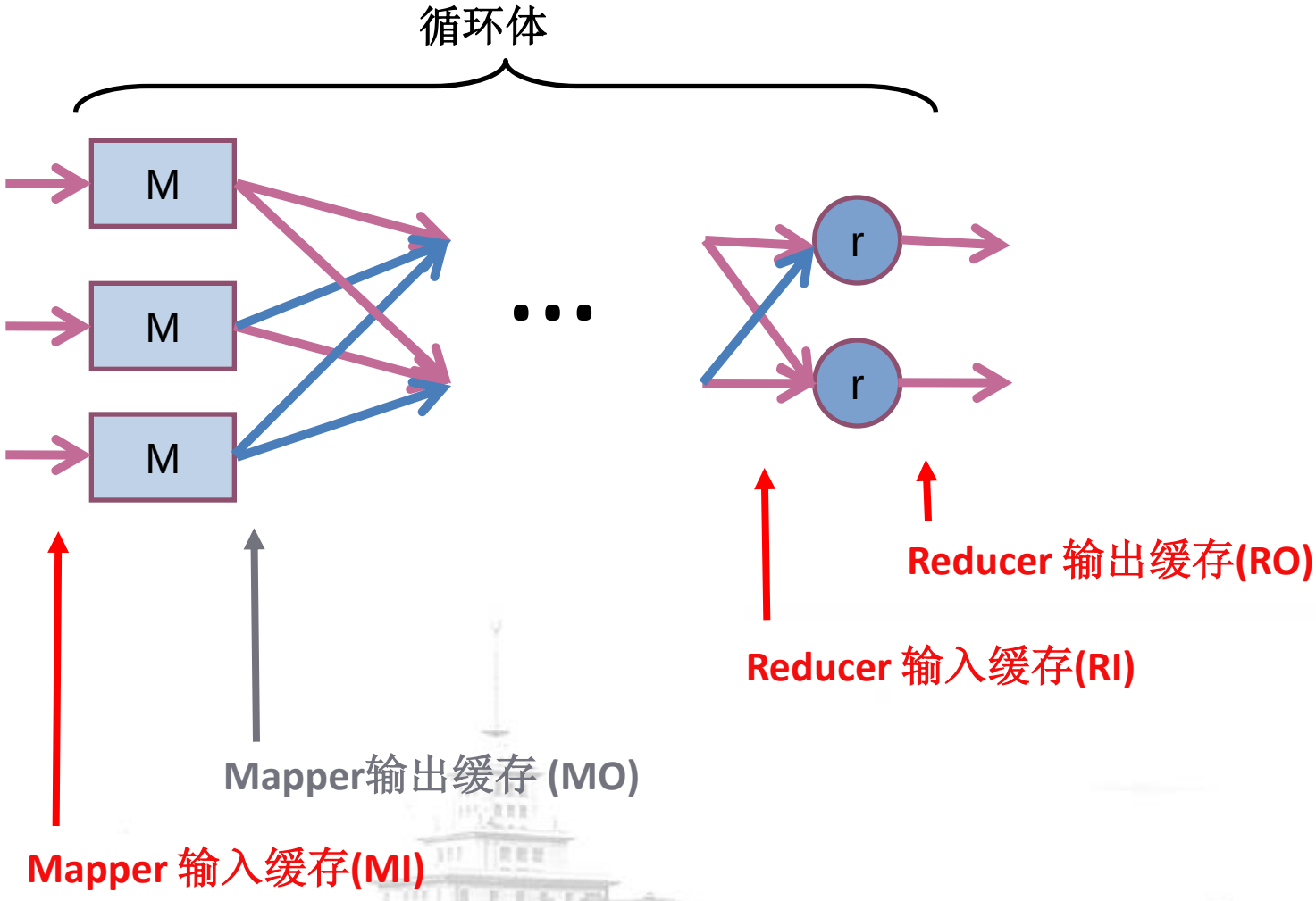
P 是循环不变量

1. 每次循环载入P

最佳化方案

- 缓存(索引)

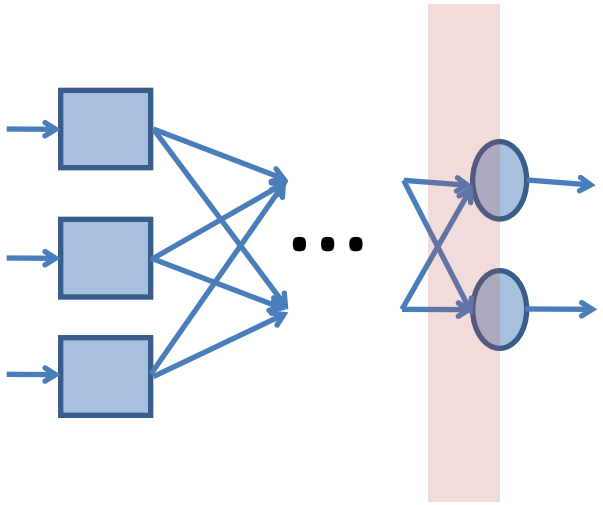
方法: 缓存迭代


循环体
M
M
M
r
r
Reducer 输出缓存(RO)
Reducer 输入缓存(RI)
Mapper输出缓存 (MO)
Mapper 输入缓存(MI)

RI: Reducer输入缓存

- 提供的功能:
  - 无需map/shuffle访问循环不点的数据
- 使用:

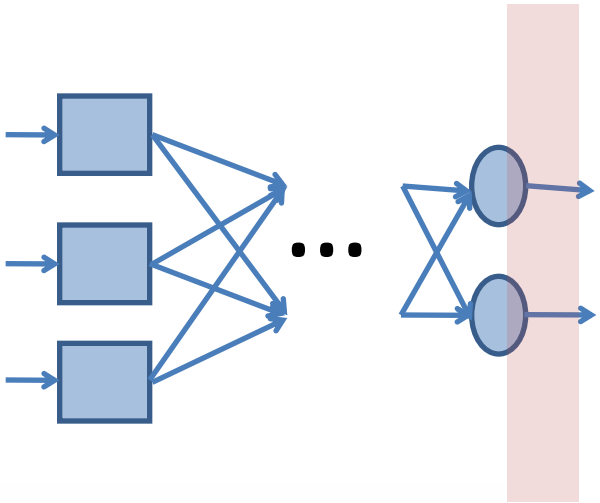
  Reducer函数
- 假设:
  1. 通过迭代, mapper输出一个给定的常数表
  2. 静态划分方法 (这意味着: 没有新的节点)



- RegeRank
  - 在每一步避免重排网络
- 传递闭包
  - 在每一步避免重排图
- K-means
  - 没有帮助

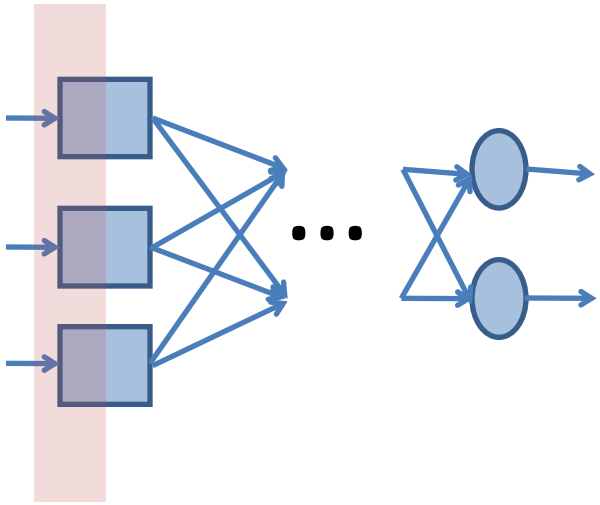
RO: Reducer输出缓存

- 提供的功能:
  - 分布式访问使用之前轮的输出结果
- 使用:
  - 不动点求值
- 假设:
  1. 通过迭代划分常量
  2. Reducer的输出键在功能上决定了Reducer的输入键
- PageRank
  - 允许分布式不动点求值
  - 排除额外的MapReduce工作
- 传递闭包
  - 没有帮助
- K-means
  - 没有帮助



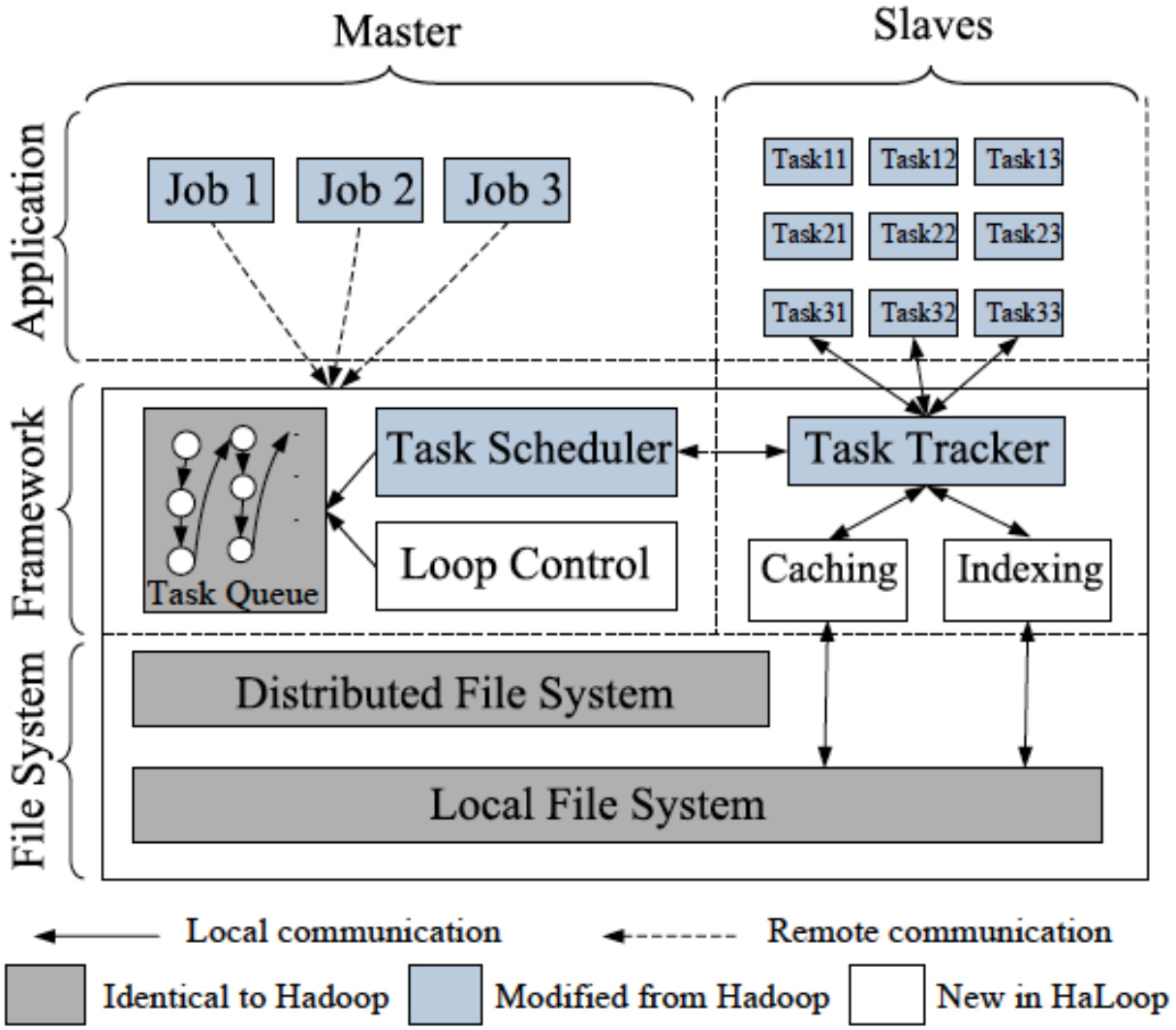
MI:Mapper输入缓存

- 提供的功能:
  - 在后面的迭代中访问非局部mapper的输入
- 使用:
  - 调度map任务
- 假设:
  1. Mapper输入不发生改变



- PageRank
  - 通过使用Reducer的输入缓存使用
- 传递闭包
  - 通过使用Reducer的输入缓存使用
- K-means
  - 在>0轮迭代避免非局部数据读入

体系结构


Master
Slaves
Application
Job 1
Job 2
Job 3
Task11
Task12
Task13
Task21
Task22
Task23
Task31
Task32
Task33
Framework
Task Queue
Task Scheduler
Loop Control
Task Tracker
Caching
Indexing
File System
Distributed File System
Local File System
Local communication
Remote communication
Identical to Hadoop
Modified from Hadoop
New in HaLoop

编程接口：支持迭代

```
Job job = new Job();

job.AddMap(Map Rank, 1);
job.AddReduce(Reduce Rank, 1);
job.AddMap(Map Aggregate, 2);
job.AddReduce(Reduce Aggregate, 2);

job.AddInvariantTable(#1);
job.SetInput(IterationInput);

job.SetFixedPointThreshold(0.1);
job.SetDistanceMeasure(ResultDistance);
job.SetMaxNumOfIterations(10);

job.SetReducerInputCache(true);
job.SetReducerOutputCache(true);

job.Submit();
```

定义循环体

声明输入变量**invariant**

定义循环体输入，以轮数为参数

停止条件

启动缓存

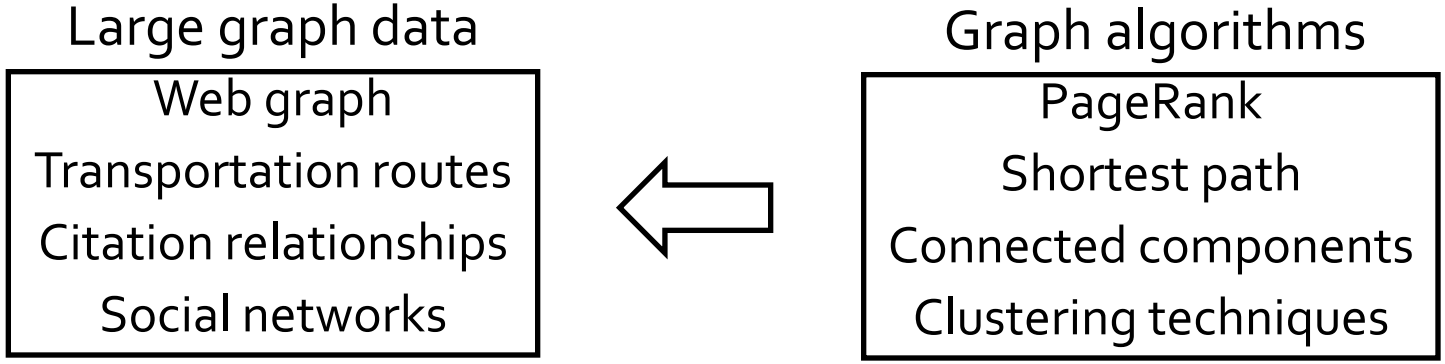
本讲内容

9.1 基于迭代处理平台的并行算法

9.2 基于图处理平台的并行算法

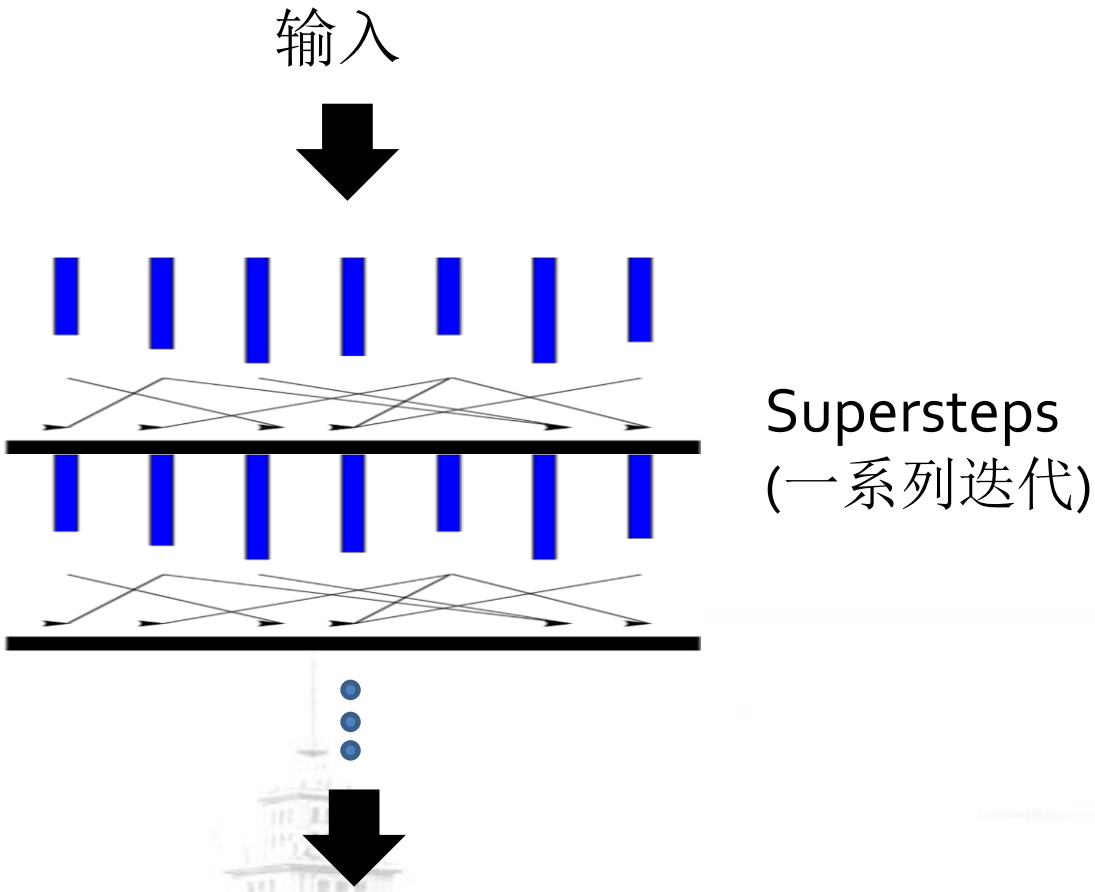
介绍

- 许多实际计算机问题会涉及到大型图



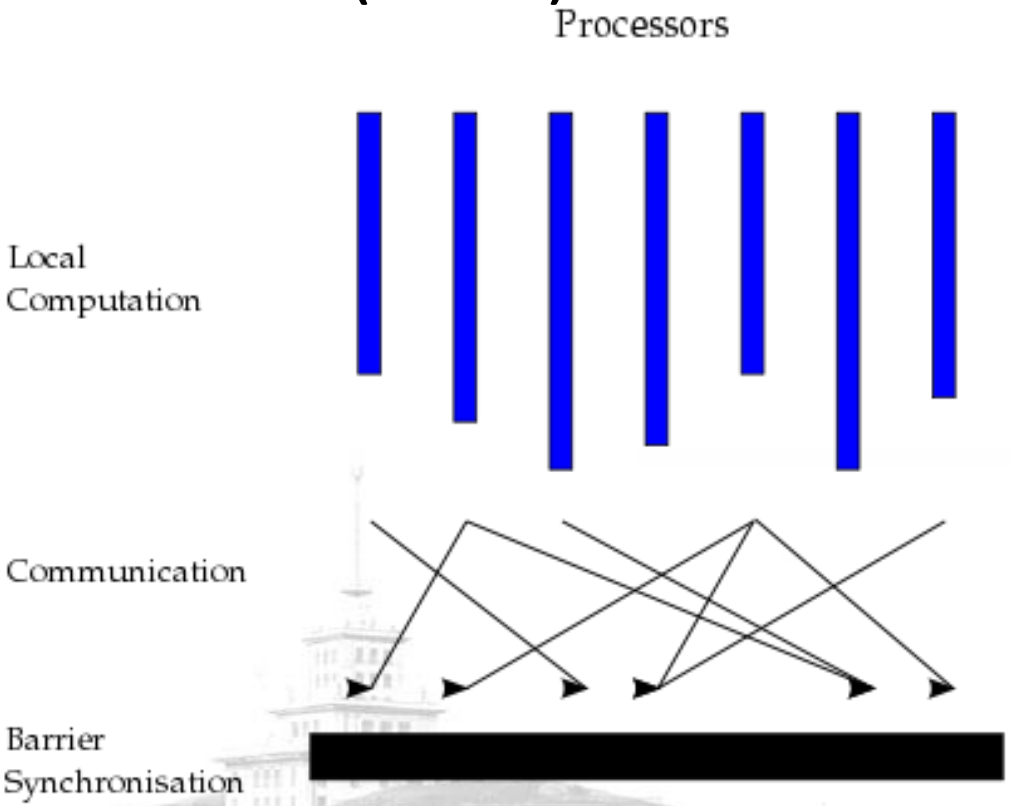
- MapReduce不适合图处理
  - 并行处理需要多次迭代，这导致MapReduce的迭代，影响到了整体的性能

计算模型(1/3)



计算模型

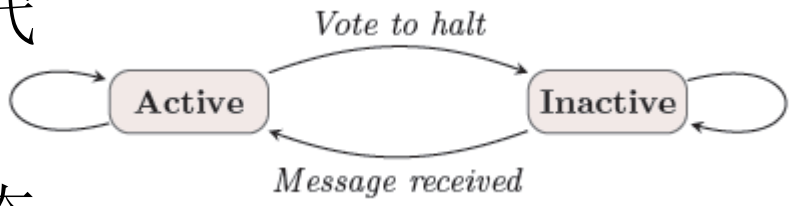
- “想象成一个顶点”
- 由Valiant’s Bulk 提出的BSP模型（Synchronous Parallel model (1990)）



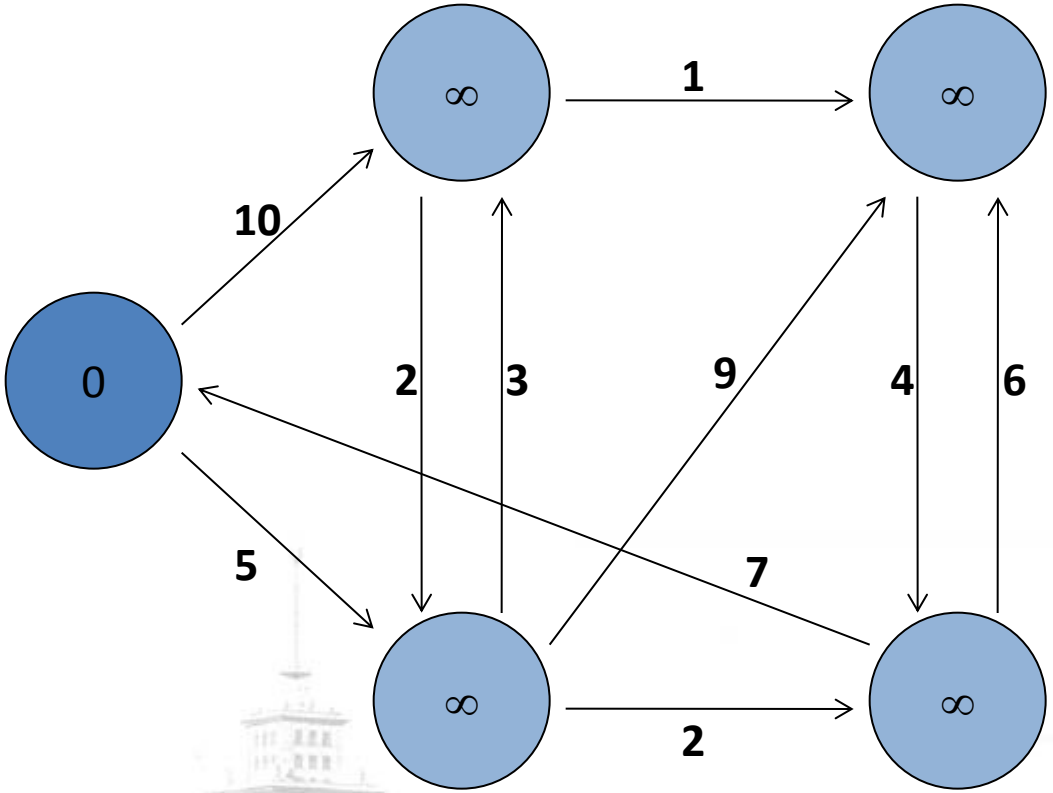
Source: http://en.wikipedia.org/wiki/Bulk_synchronous_parallel

计算模型

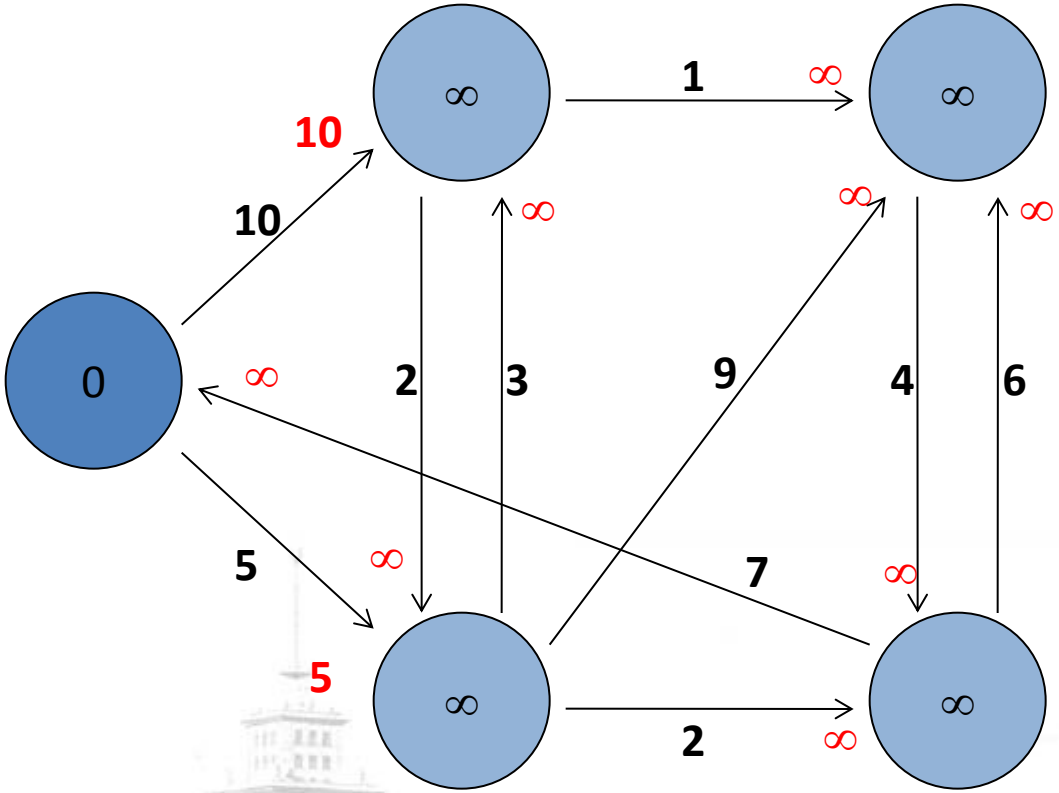
- Superstep: 并行结点计算
  - 对于每个结点
    - 接受上一个superstep发出的消息
    - 执行相同的用户定义函数
    - 修改它的值或者其输出边的值
    - 将消息送到其他点(由下一个superstep接受)
    - 改变图的拓扑结构
    - 没有额外工作要做时结束迭代
  - 终止条件
    - 所有顶点同时变为非活跃状态
    - 没有信息传递



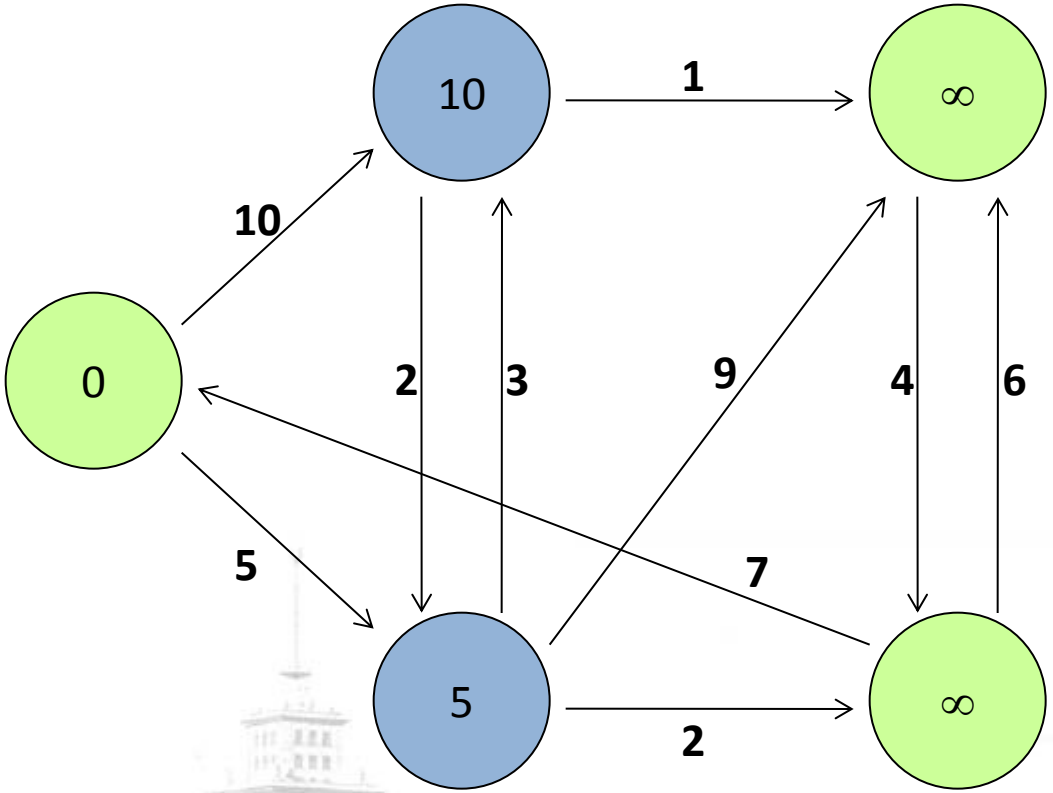
例: SSSP – Pregel并行广度优先搜索



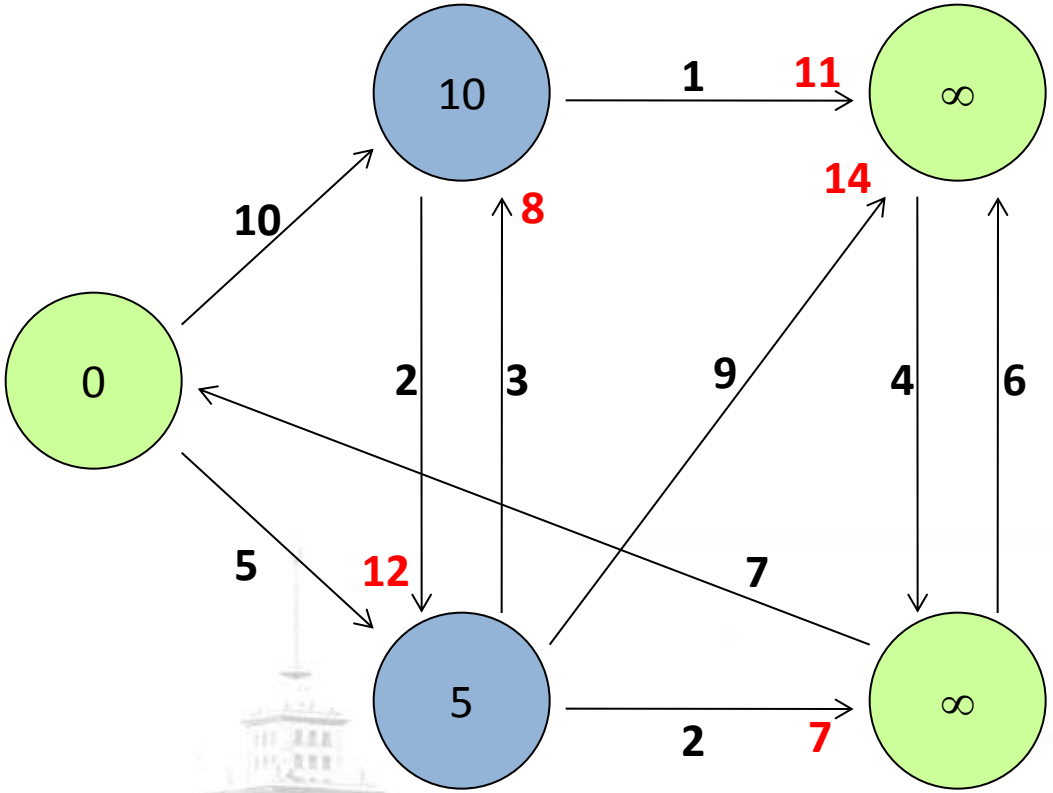
例: SSSP – Pregel并行广度优先搜索



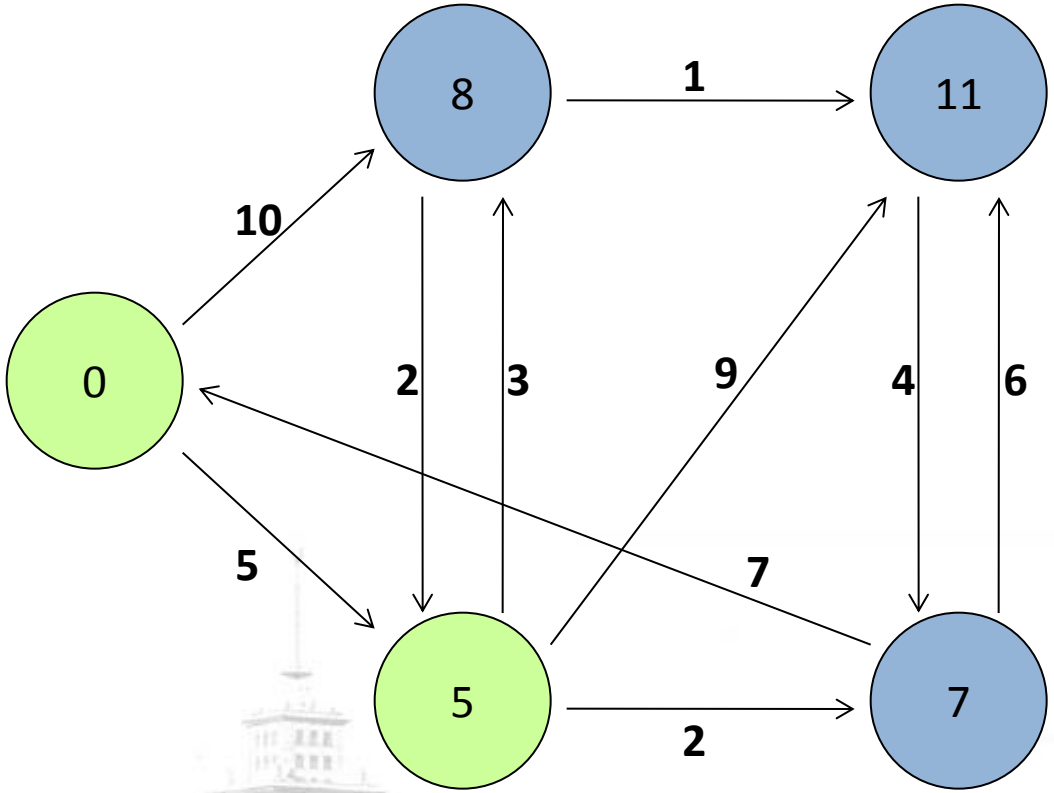
例: SSSP – Pregel并行广度优先搜索



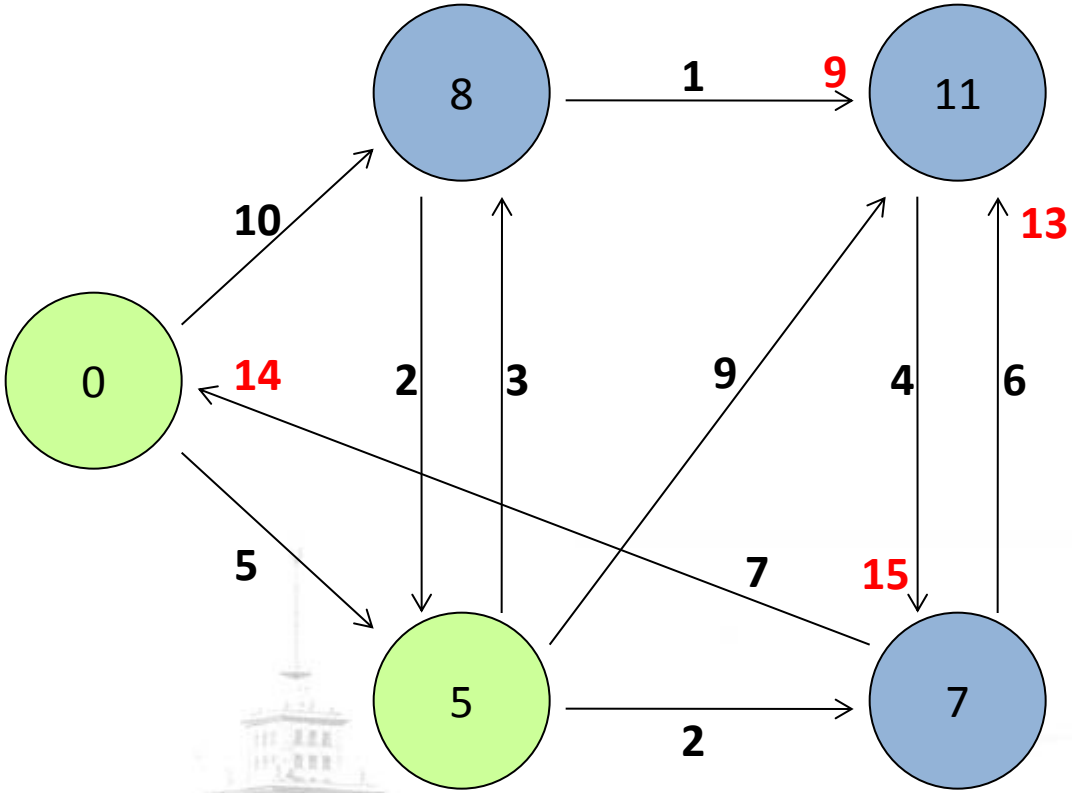
例: SSSP – Pregel并行广度优先搜索



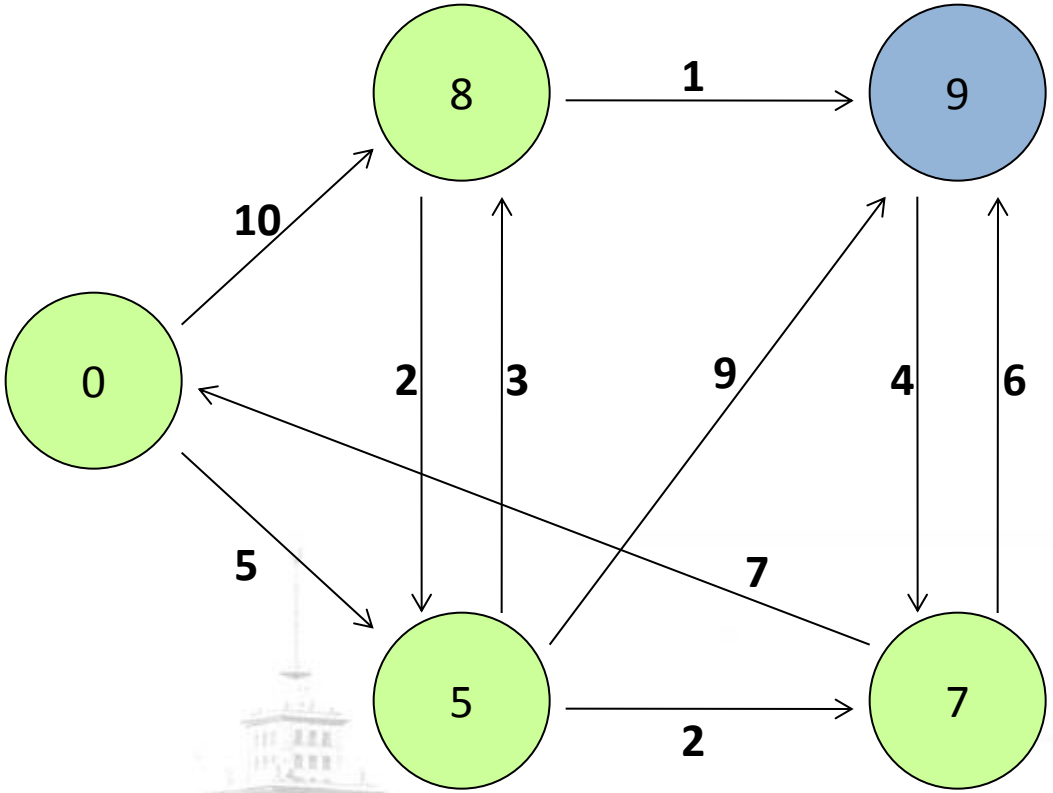
例: SSSP – Pregel并行广度优先搜索



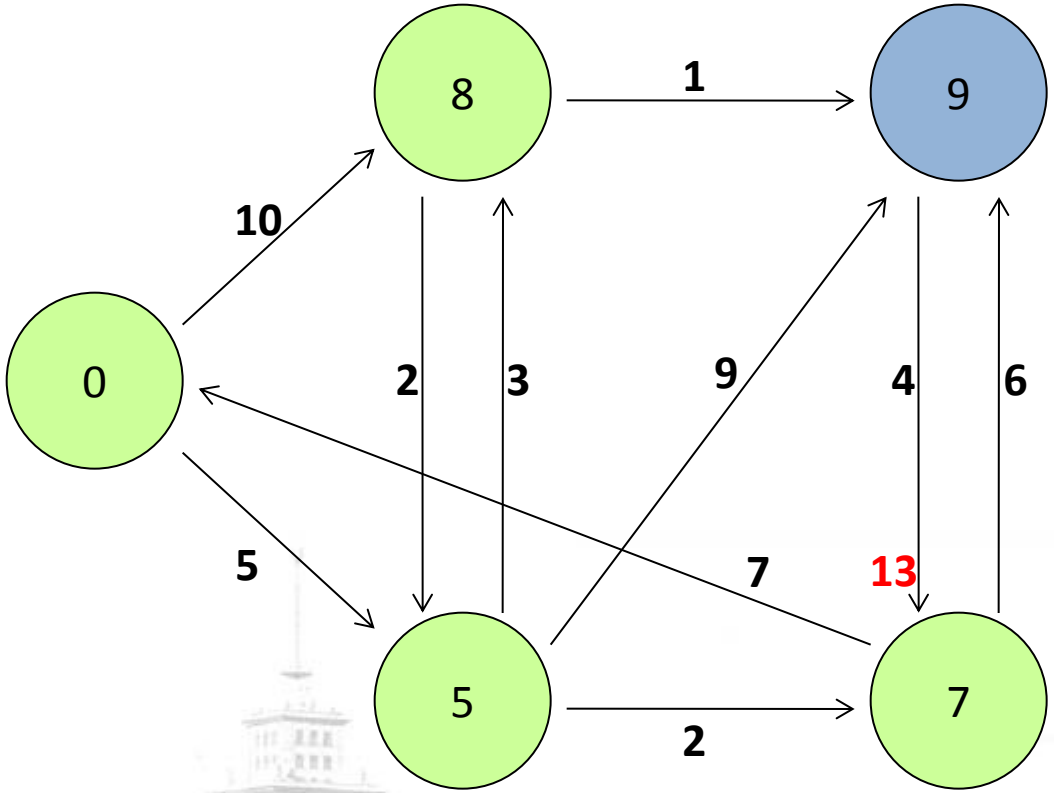
例: SSSP – Pregel并行广度优先搜索



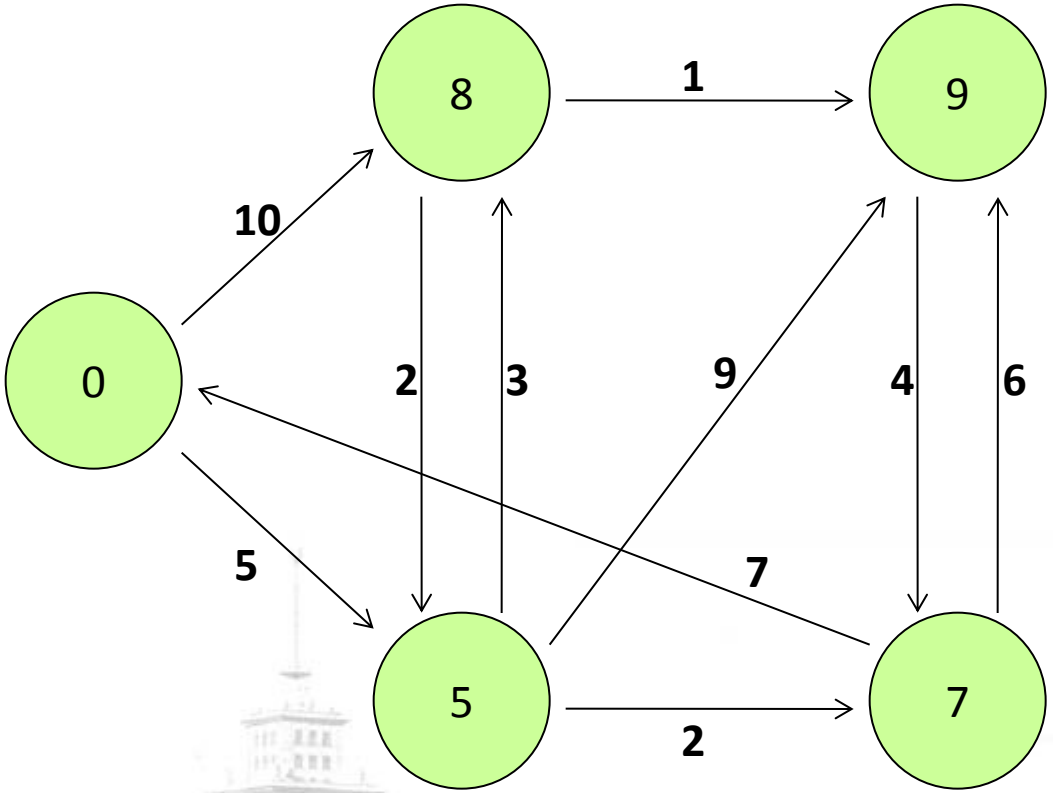
例: SSSP – Pregel并行广度优先搜索



例: SSSP – Pregel并行广度优先搜索



例: SSSP – Pregel并行广度优先搜索



与MapReduce的不同之处

- 图算法可以被写成一系列的MapReduce调用
- Pregel
  - 在执行计算的机器上保持顶点和边
  - 用网状结构传输信息

- MapReduce
  - 每一阶段都利用整个图的全部状态
  - 需要整合MapReduce链

C++ API

- 编写一个Pregel程序
- 将之前定义的顶点类归入子类

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

重载!

输入msgs

输出msg

例:SSSP不动点类

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

系统安装

- Pregel系统使用master/worker进程
  - Master
    - 维护worker
    - 恢复workers产生的错误
    - 提供Web-UI 提供Web-UI监督工作进程工具
  - Worker
    - 处理自己的任务
    - 和其他workers交流

- 连续的数据以文件的格式存储在分布式存储空间系统(例如GFS 、 BigTable)
- 临时数据存储在磁盘中

Pregel程序的执行

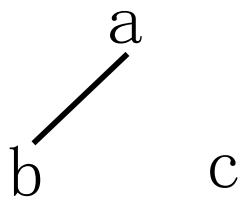
1. 多个程序的副本在服务器集群中执行
2. master分配划分内容作为每一个worker的输入
   – 每一个worker负载多个顶点并把他们标记
3. master命令每个worker执行superstep
   – 每个worker循环通过已标记的顶点计算每个顶点
   – 信息异步传输, 但都在superstep结束前传送
   – 重复以上步骤知道没有已标记顶点或信息传输
4. 程序运行停止后,master会命令每一个worker保存图的部分信息

例子：计算子图同构

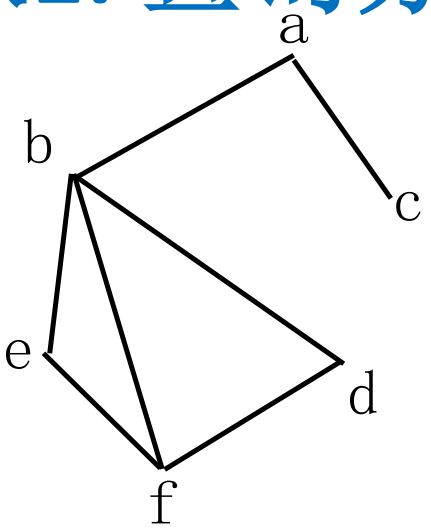
- 问题：给定(结点有标签的)数据图G和查询图P，找到G中和P同构的子图。

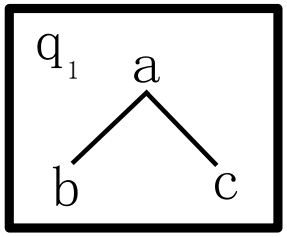
方法概述

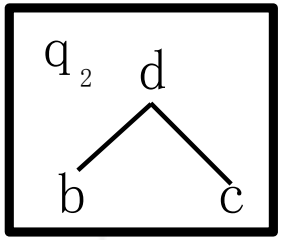
- 基本思想
  - Join作为基本框架
  - 利用搜索避免无用的中间结果
- 基本单位: STwig
- 三个步骤
  - 查询分解
  - 搜索
  - Join

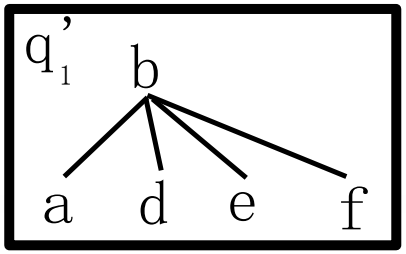


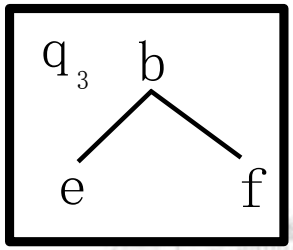
步骤1: 查询分解

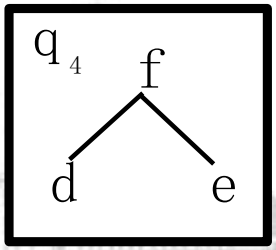


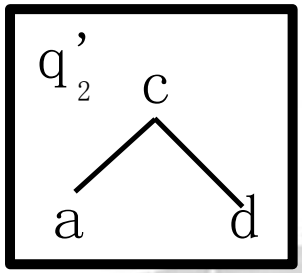


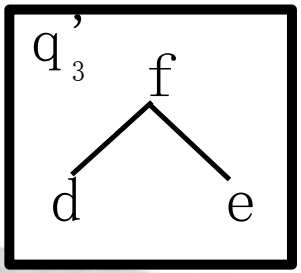




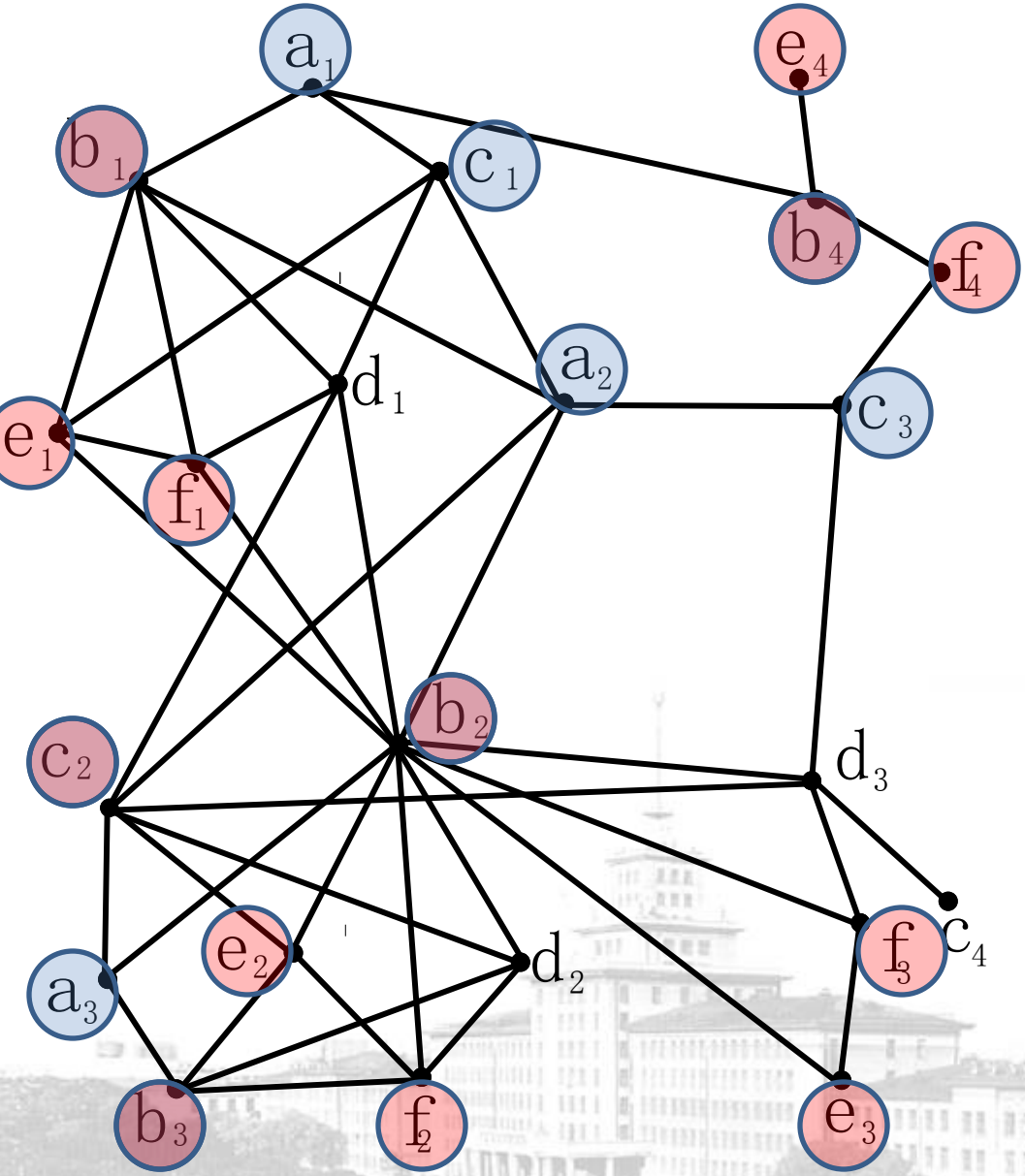


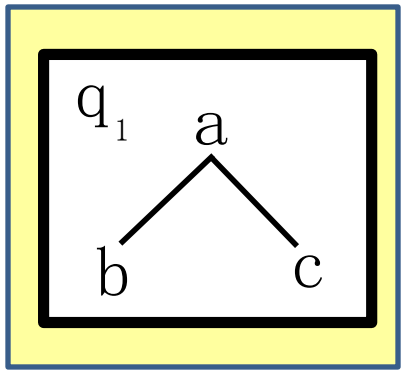


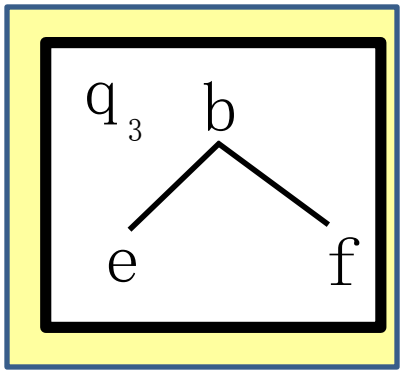




步骤2: 搜索







H_a {a_1, a_2, a_3}

H_b {b_1, b_2, b_3 , b_4}

H_c {c_1, c_2, c_3}

Step 3: Join

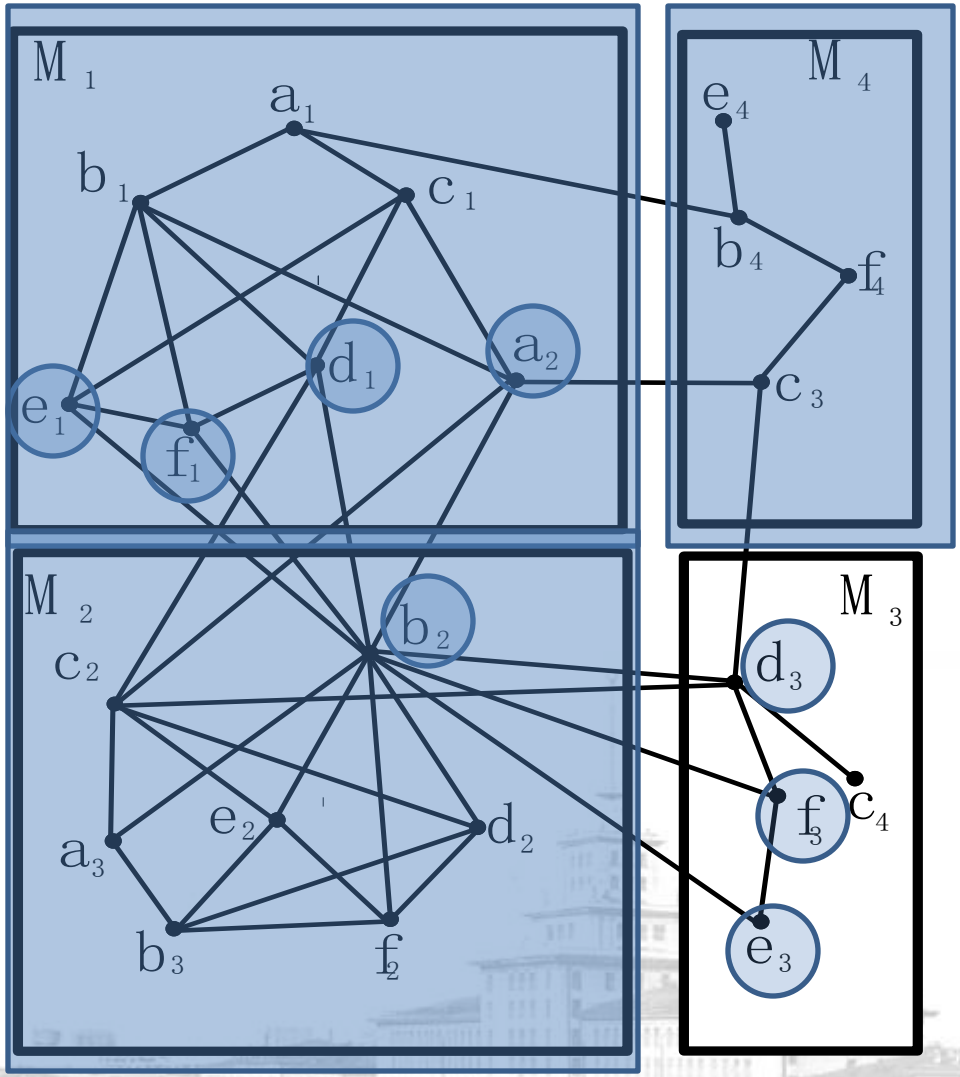
- 连接中间结果生成最终结果

(a_1, b_1, c_1)

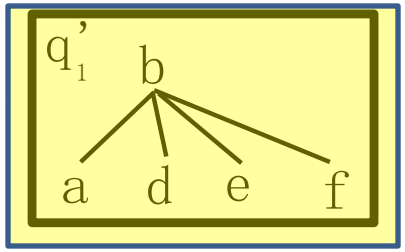
(b_1, e_1, f_1)

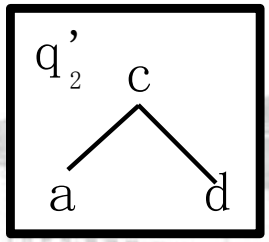
$\Rightarrow (a_1, b_1, c_1, e_1, f_1)$

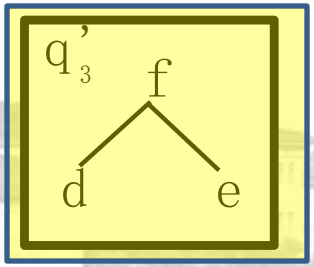
并行子图匹配



- 模式分解
- 搜索
  - 加载邻居
- Join
  - 加载需要的中间结果







致谢

- 本讲义部分内容来自于Bill Howe的讲义