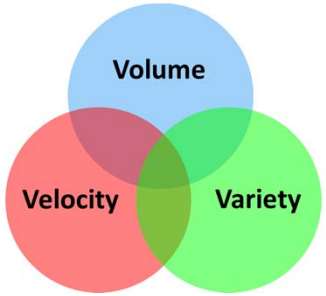
大数据系统与大规模数据分析

大数据运算系统（3）



陈世敏

中科院计算所
计算机体系结构
国家重点实验室

©2015-2018 陈世敏

Outline

- GraphLab
- PowerGraph

异步图运算系统

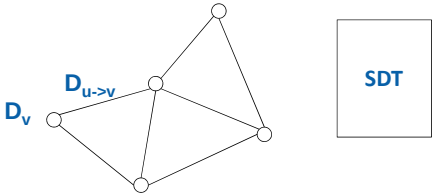
- 许多机器学习算法可以用图运算来实现
- 但是同步图运算需要进行多次的迭代
- 思路：异步图运算
  - 允许不同顶点有不同的更新速度
  - 一个顶点的更新，它的邻居顶点立即可见，而不是等到下一个超步开始
  - 从而可以更快速地收敛
    - 注意：许多机器学习算法没有“精确”的结果
    - 异步计算收敛的结果和同步计算收敛的结果可能是不一样的

异步图运算系统 GraphLab

- “***GraphLab: A New Framework For Parallel Machine Learning***”. Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, Joseph M. Hellerstein. **UAI 2010**.
- “*Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud*.” Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin and Joseph M. Hellerstein. **PVLDB 2012**.

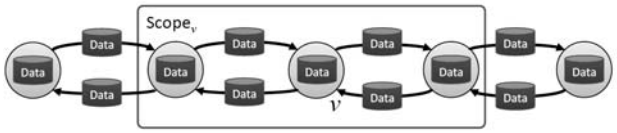
数据模型

- Data graph G=(V, E)
  - 每个顶点可以有数据D_v
  - 每条边可以有数据$D_{u->v}$
- 全局数据表(SDT, shared data table)
  - SDT[key] ➔ value
  - 可以定义全局可见的数据



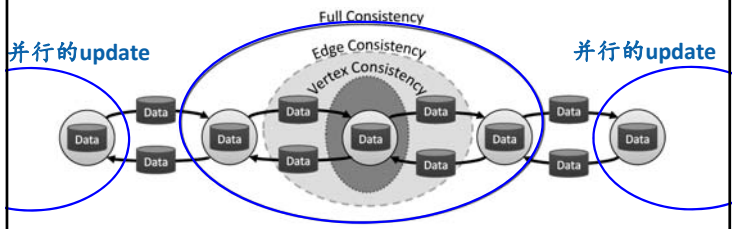
顶点计算

- D_{Scopev} = update (D_{Scopev} ;SDT)
  - update类似Pregel compute，是程序员定义的顶点运算
  - $Scope_v$是顶点运算涉及的范围
    - 包括顶点v，v的相邻边，和v的相邻顶点
  - 运算是**直接访问内存**进行的
  - update直接修改顶点和边上的数据，**修改立即可见**，而不是下个超步才可见



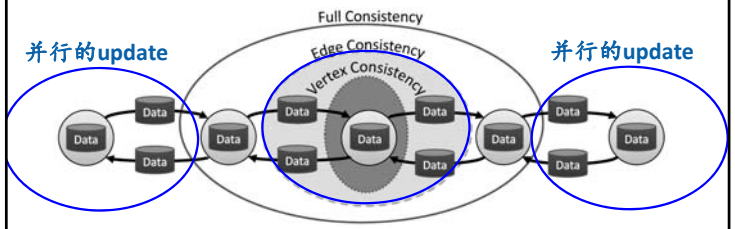
三种一致性模型

- Full consistency: 系统保证在update执行过程中，没有任何其它函数会访问$Scope_v$
  - 这种模型对并行执行的限制最大
  - 但是任何update都能够正确执行



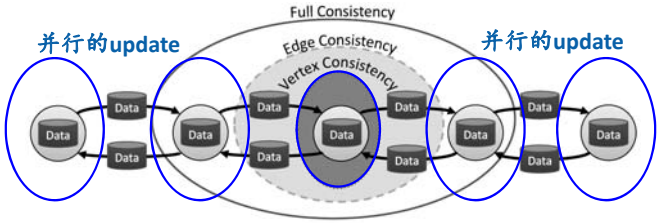
三种一致性模型

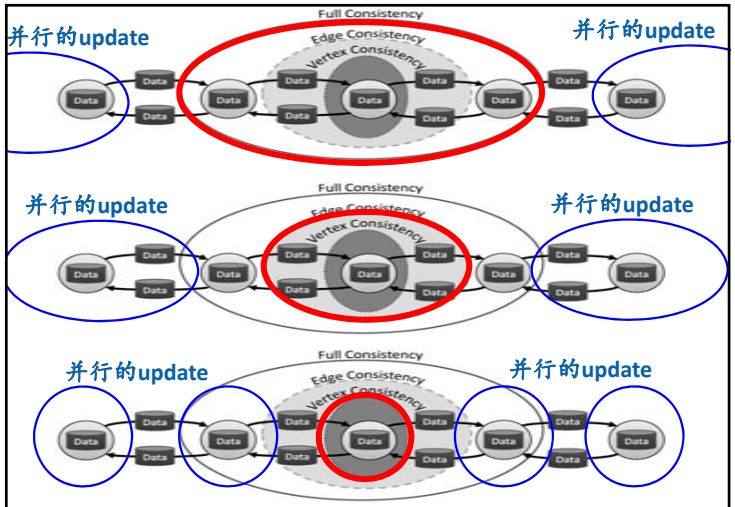
- Edge consistency: 系统保证在update执行过程中，没有任何其它函数会访问v的Edge和v本身
  - update不能写邻居顶点的数据，但是可以读邻居顶点的数据



三种一致性模型

- Vertex consistency: 系统保证在update执行过程中，没有任何其它函数会访问v本身
  - Update只能够读写本顶点的数据，和读v邻边的数据





Scheduling

- 按照什么顺序访问顶点调用update?
- 例如：
  - Synchronous scheduler: 类似同步图运算
  - Round-robin scheduler: 顺序计算每个顶点，下一个顶点可以看到前面的计算结果
  - FIFO scheduler: update中调用AddTask创建新的Task，Task对应顶点的update，创建的Task按照先进先出顺序执行
  - Prioritized scheduler: 创建Task并指定优先顺序

Sync计算

- 除了update计算，GraphLab还定义了一种全局的计算sync
- Sync类似在所有的顶点上计算一个aggregate
  - 程序员提供fold和apply函数，给定一个初始值initial_value和一个key
  - 系统执行下面的操作：

```
t = initial_value;
foreach v ∈ V {
   t = fold(v, t);
}
SDT[key] = apply(t);
```

- 例如：可以计算update运算是否已经收敛

GraphLab

- 以顶点为中心的计算
- 异步计算
  - 可以定义不同的scheduling策略
- 可以立即看到完成的计算结果
  - 共享内存方式编程，而不是消息传递方式
  - 需要一致性模型
    - Full, edge, vertex consistency模型
- 采用一种全局的aggregate机制帮助判断是否收敛

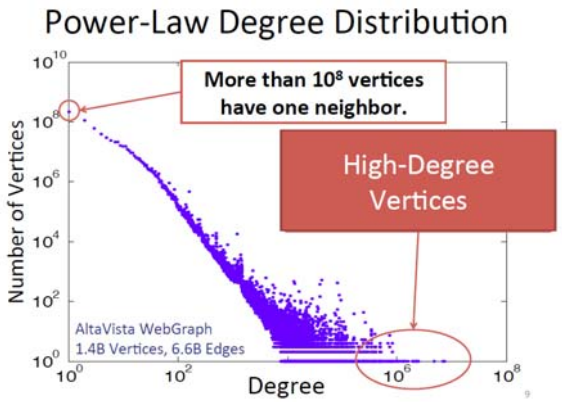
Outline

- GraphLab
- PowerGraph

PowerGraph

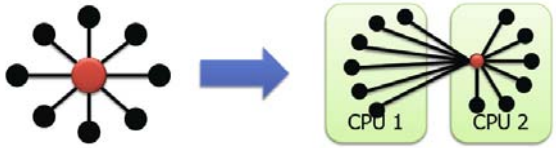
- "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs." Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. **OSDI 2012**.
- 下面的slides基于OSDI'12

大量自然图符合Power-Law



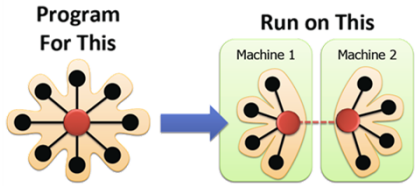
Power-Law引起的问题

- 图划分之间有大量的跨边
- Pregel
  - 需要传输大量的消息
- 如何优化？



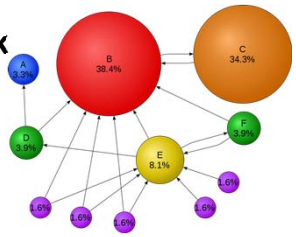
PowerGraph

- 把单一的Compute()分成三个用户函数GAS来实现
  - Gather
  - Apply
  - Scatter
- 利用了一大类图计算算法的特点
- 实际的效果
  好像把大度顶点
  分裂成为了多个



图算法举例：PageRank

- $R_u = 0.15 + 0.85 \sum_{v \in B(u)} \frac{R_v}{L_v}$
  - R_v: 顶点v的PageRank*N
  - L_v: 顶点v的出度（出边的条数）
  - $B(u)$: 顶点u的入邻居集合
  - damping factor为0.85
  - N: 总顶点个数



图来源：Wikipedia

- 计算方法
  - 初始化：所有的顶点的PageRank为1
  - 迭代：用上述公式迭代直至收敛

GraphLite的实现

```
double sum= 0.0;
for (; !msgs->done(); msgs->next()) {
    sum += msgs->getValue();
}
```
接收消息

```
val = 0.15 + 0.85 * sum;
*mutableValue() = val;
```
更新状态

```
int64_t n = getOutEdgeIterator().size();
sendMessageToAllNeighbors(val / n);
```
发送消息

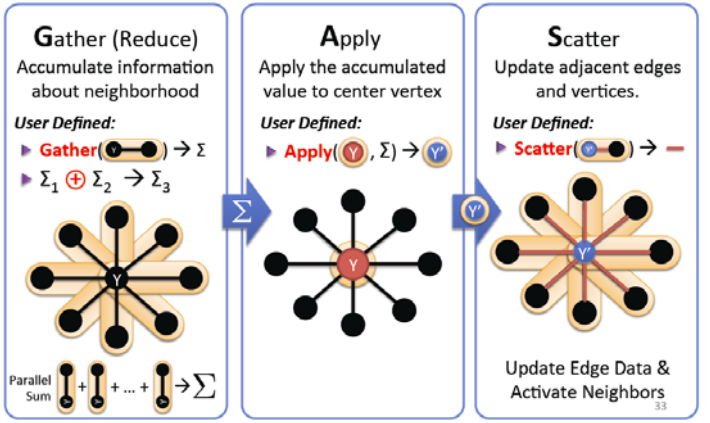
GAS抽象

```
double sum= 0.0;
for (; !msgs->done(); msgs->next()) {
    sum += msgs->getValue();
}
```
改为 Gather

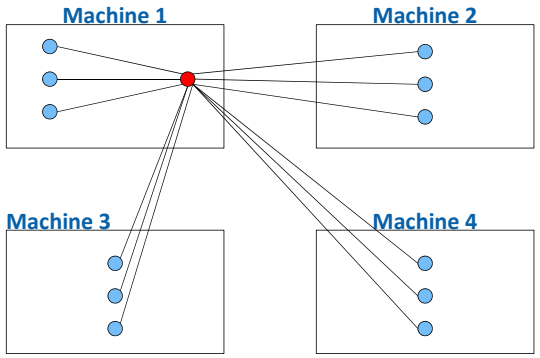
```
val = 0.15 + 0.85 * sum;
*mutableValue() = val;
```
改为 Apply

```
int64_t n = getOutEdgeIterator().size();
sendMessageToAllNeighbors(val / n);
```
改为 Scatter

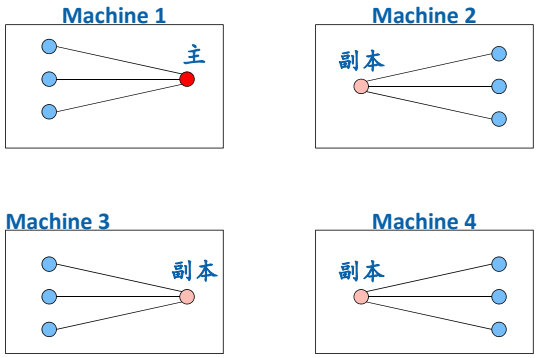
GAS Decomposition



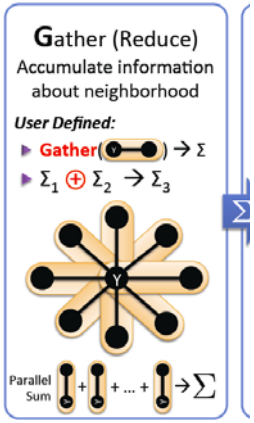
具体实现：通常的划分



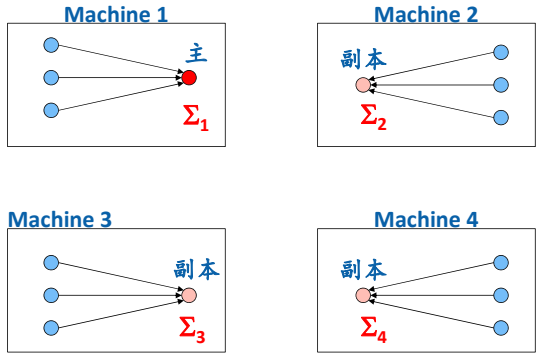
具体实现：PowerGraph



GAS Decomposition

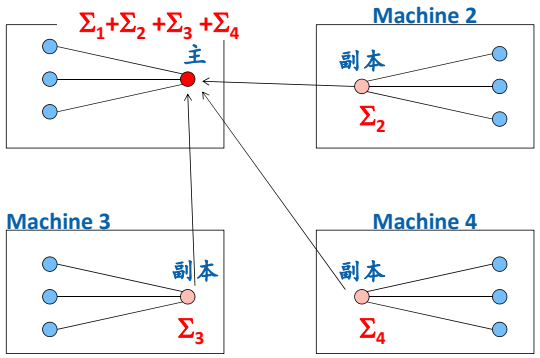


具体实现：Gather，收消息求局部和



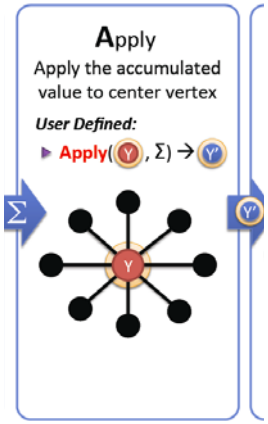
- Gather( ) → Σ
- Σ1 ⊕ Σ2 → Σ3

具体实现：Gather，局部和➔全局和

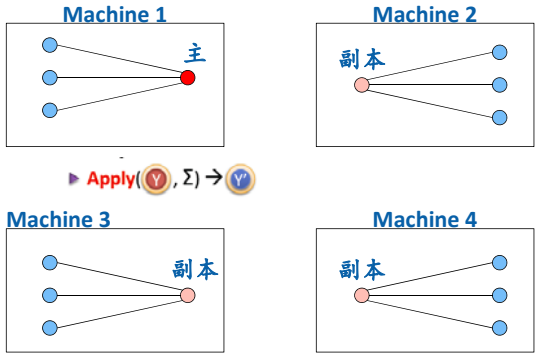


- Gather( ) → Σ
- Σ1 ⊕ Σ2 → Σ3

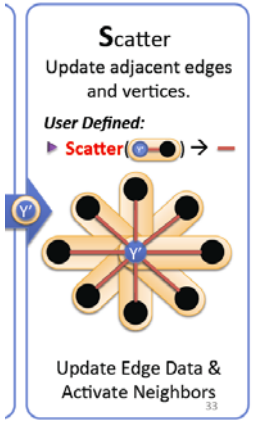
GAS Decomposition



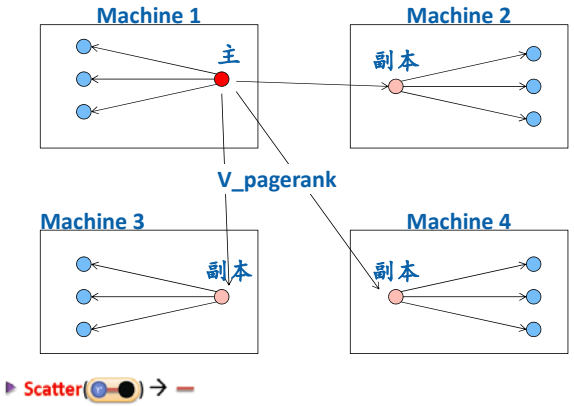
具体实现：Apply，在主顶点更新PageRank



GAS Decomposition



具体实现：Scatter，发送消息



小结

- Pregel, GraphLite
- GrapLab
- PowerGraph