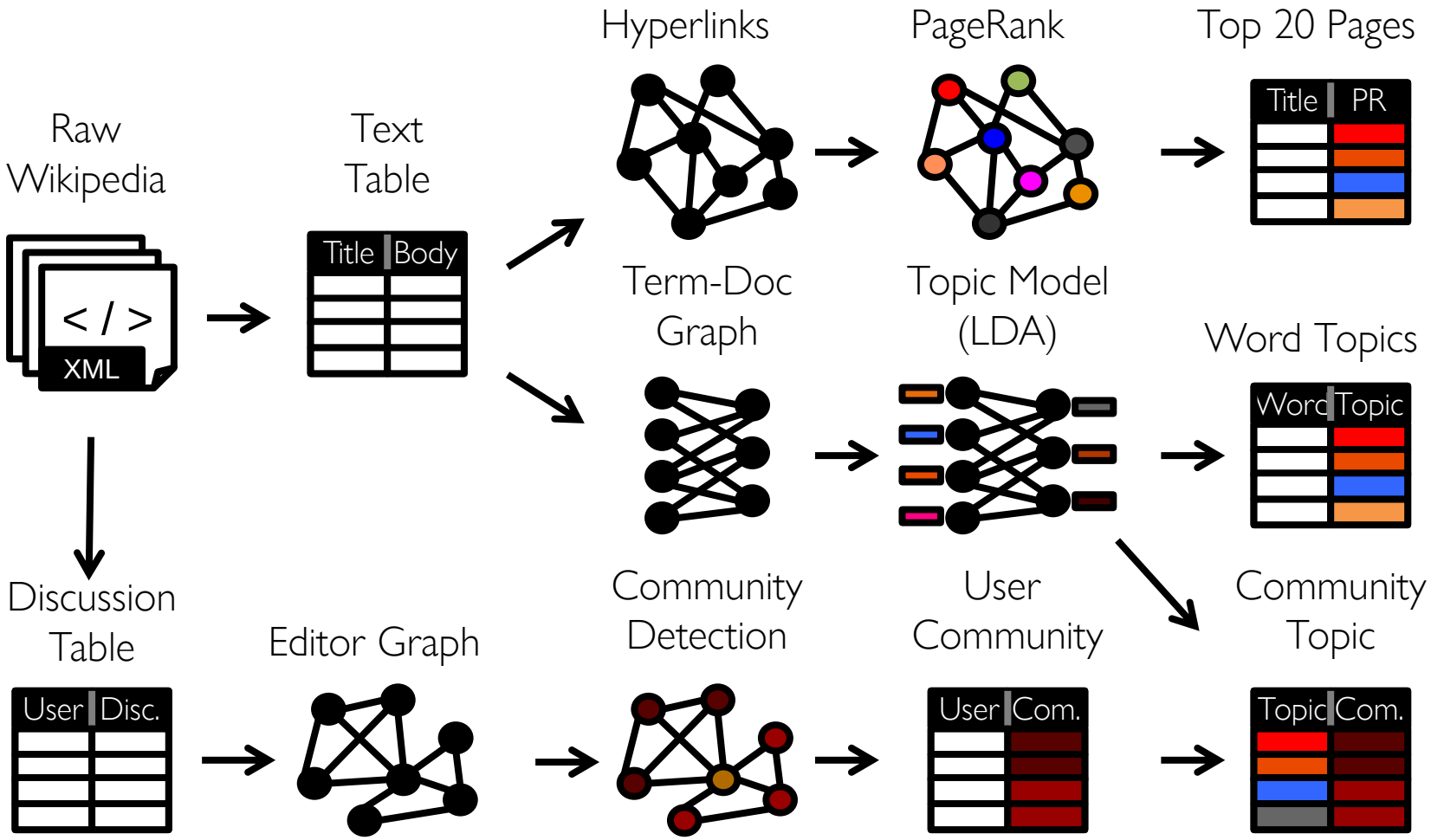
第九讲 图计算

Graph Processing Frameworks

目录

一、什么是图计算

图计算：基于图的分析与计算



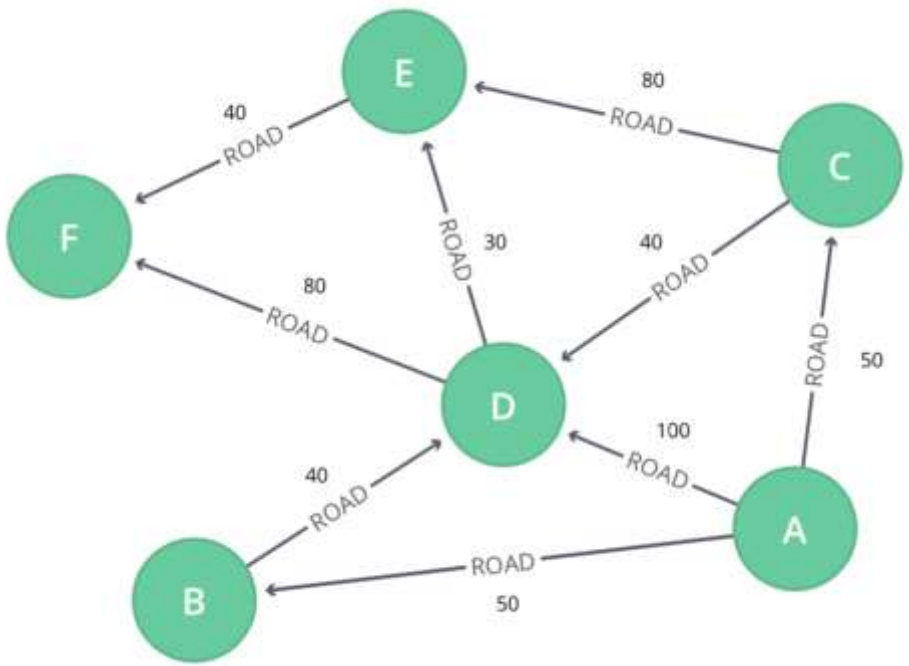
图计算问题举例

例如：寻找两地之间最短路径，网络路由的优化等

例如：寻找社交网络中的最有影响力人物，通信网络或电力网络中的易受攻击节点等。

例如：对商品网络进行聚类分析、从罪犯关系网络中锁定犯罪团伙等。

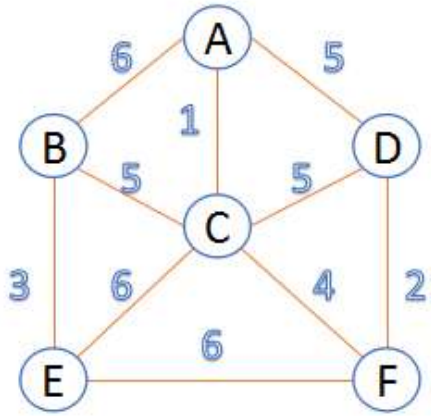
路径搜索举例： 最短路径算法



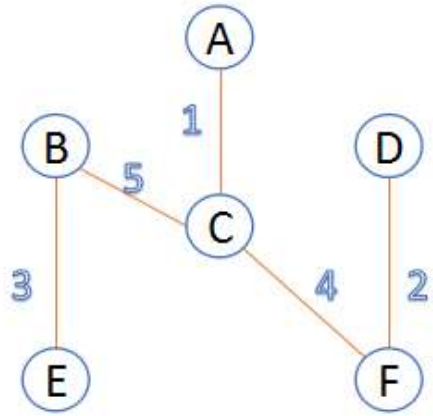
- 迪杰斯特拉算法（Dijkstra算法）
- 弗洛伊德算法（Floyd算法）
- SPFA算法

路径搜索举例： 最小生成树-Minimum Weight Spanning Tree

- 基本概念：
  - 连通图：在无向图中，若任意两个顶点vi与vj都有路径相通，则称该无向图为连通图。
  - 强连通图：在有向图中，若任意两个顶点vi与vj都有路径相通，则称该有向图为强连通图。
  - 生成树：一个连通图的生成树是指一个连通子图，它含有图中全部n个顶点，但只有足以构成一棵树的n-1条边。一颗有n个顶点的生成树有且仅有n-1条边，如果生成树中再添加一条边，则必定成环。
  - 最小生成树：在连通网的所有生成树中，所有边的代价和最小的生成树，称为最小生成树。
- 常见应用：
  - 矿井通风管道设计问题
  - 城市之间怎么修路可以使整体上路最短
  - 城市之间构建通信网络怎样可以使整体的话费最少



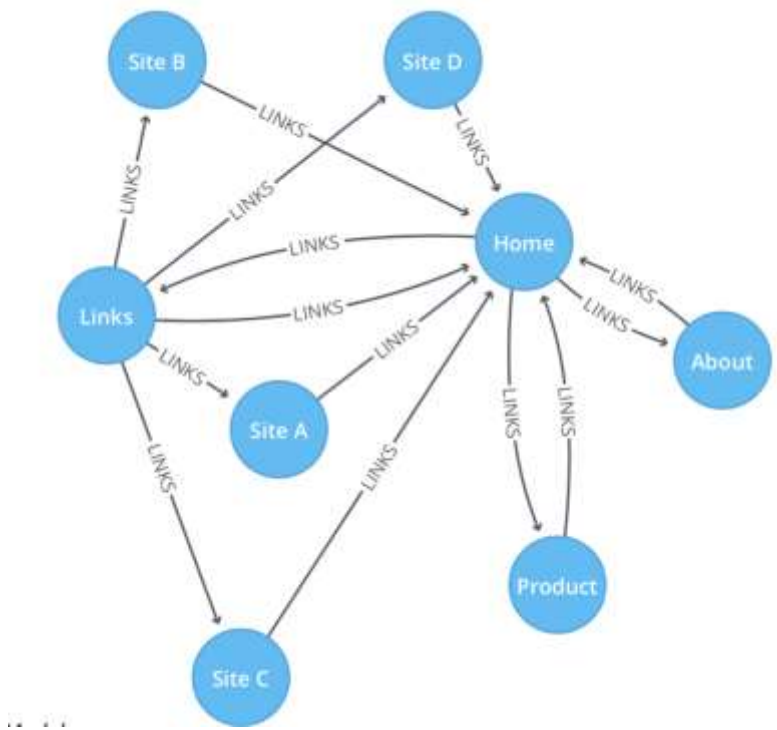
连通网G



最小生成树

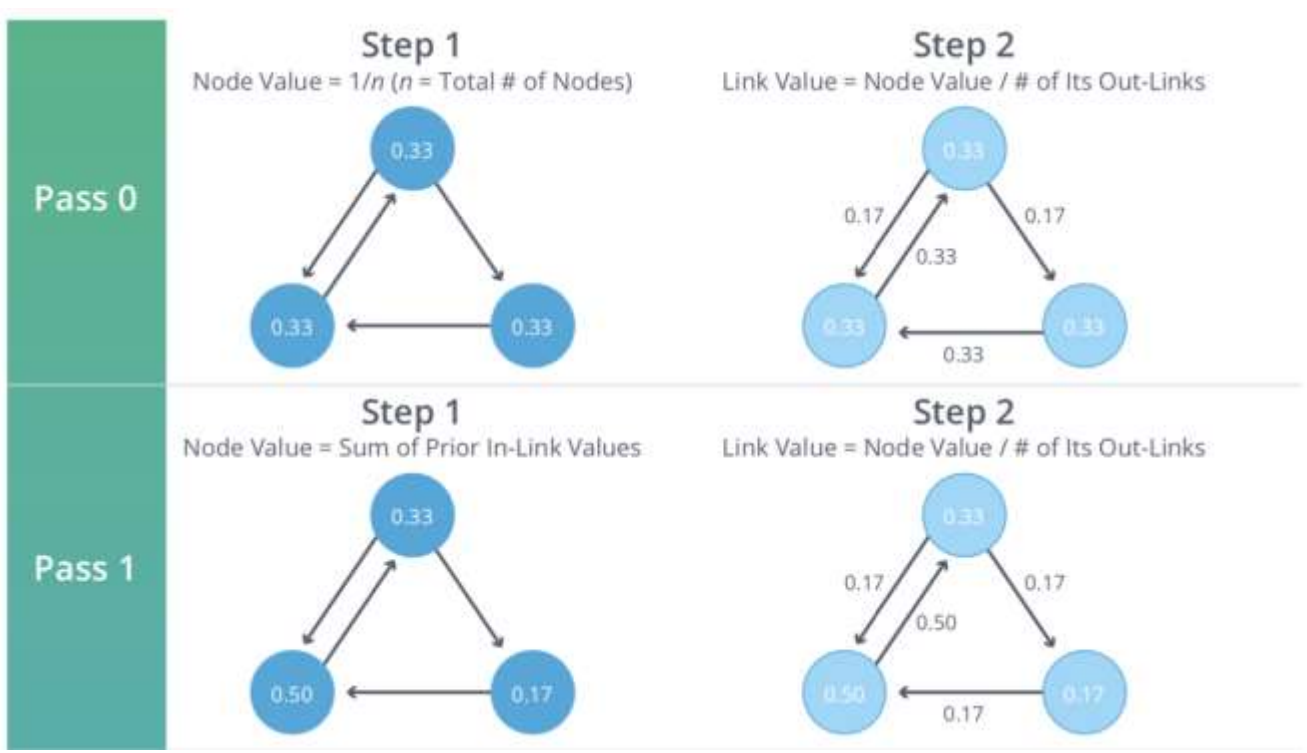
- Prim算法
- Kruskal算法
- Sollin算法

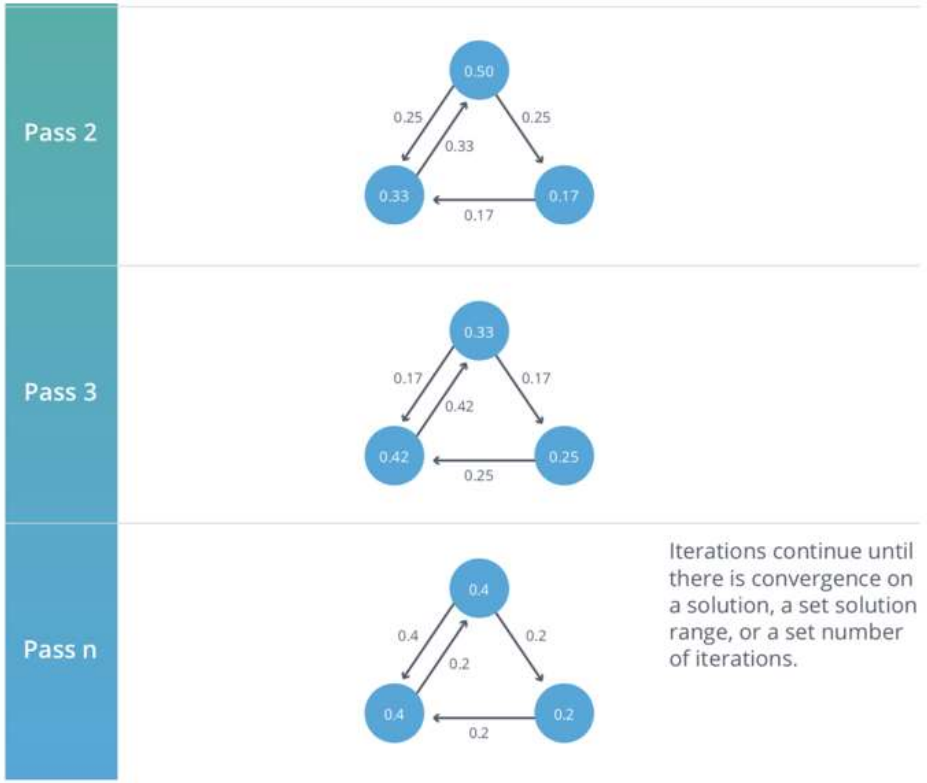
Centrality Algorithms: PageRank



PageRank计算到页面链接的数量和质量，从而确定对该页面的重要性的估计。 基本假设是，重要页面更有可能收到来自其他有影响力页面的大量链接。例如，相比拥有许多低影响力的朋友，拥有几个有影响力的朋友可以增加您的PageRank。

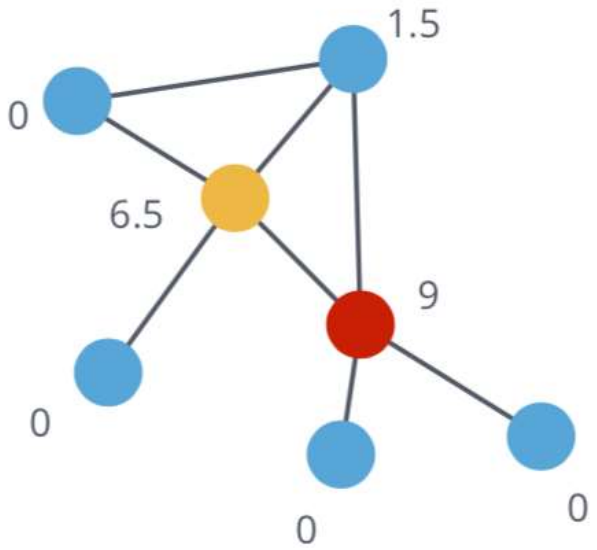
Centrality Algorithms: Pagerank





Centrality Algorithms: Betweenness Centrality

中介中间度是一种检测节点对图形中信息流的影响程度的方法。 它通常用于查找充当从图的一部分到另一部分的桥梁的节点。

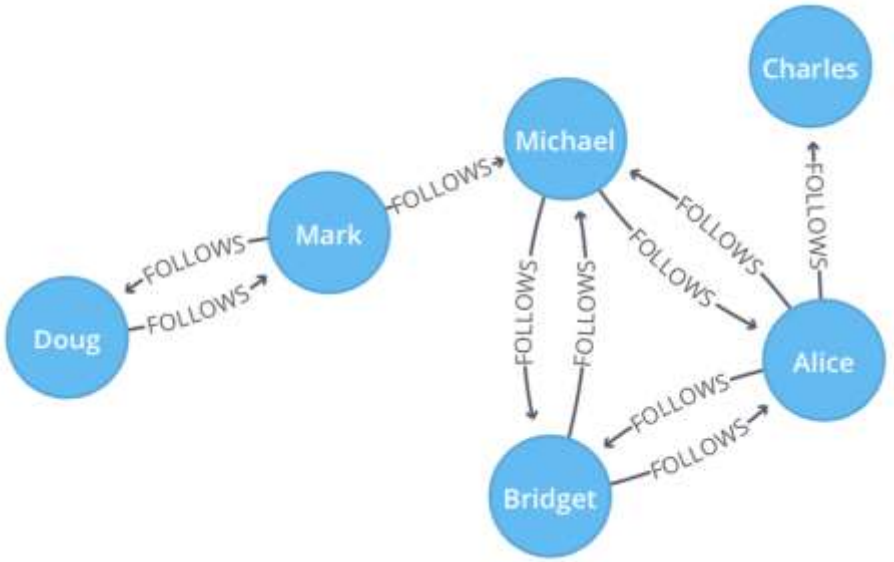


Betweenness Centrality

- 首先，找到所有最短路径
- 然后，对于每个节点，将通过该节点的最短路径数除以图中的最短路径总数
- 得分越高具有高的中间度（红色节点最高，然后是黄色节点）

社区发现算法： Strongly Connected Components

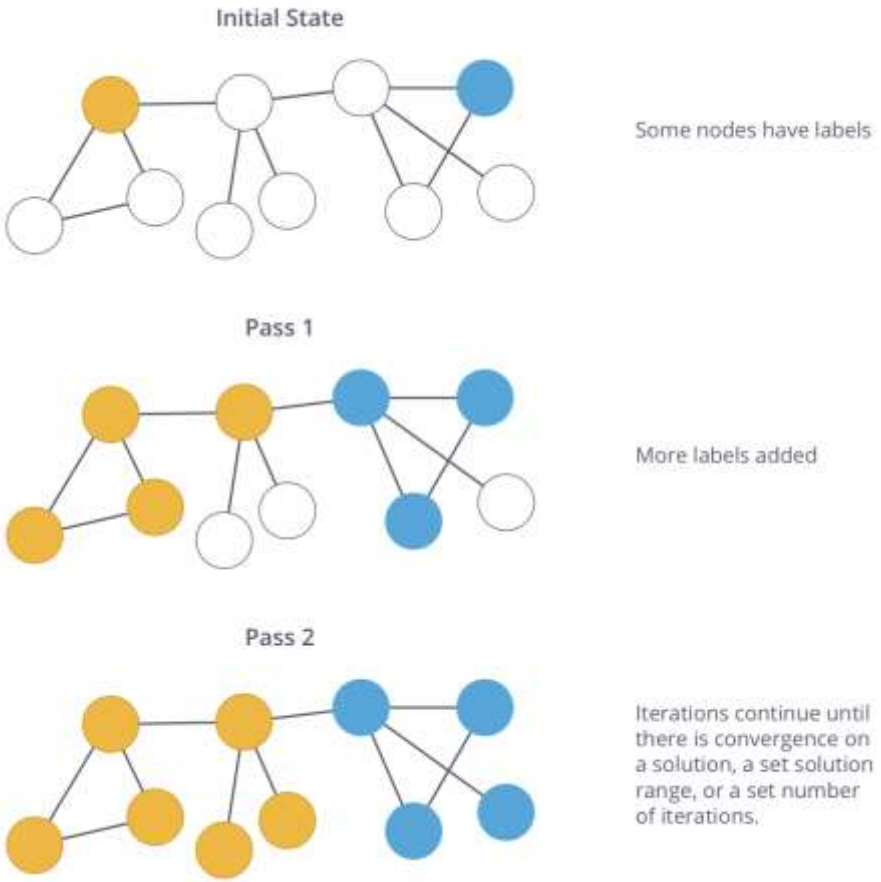
- 在有向图G中，如果两个顶点vi,vj间（vi>vj）有一条从vi到vj的有向路径，同时还有一条从vj到vi的有向路径，则称两个顶点强连通(strongly connected)。
- 如果有向图G的每两个顶点都强连通，称G是一个强连通图。
- 有向图的极大强连通子图，称为强连通分量(strongly connected components)。



Korasaju算法

Tarjan算法

社区发现算法：Label Propagation



标签传播算法（LPA）是一种用于在图形中查找社区的快速算法，该算法的直觉是单个标签可以在密集连接的节点组中迅速占主导地位，但在穿越稀疏连接区域时会遇到麻烦 。 标签将被困在一个密集连接的节点组中，并且在算法完成时以相同标签结尾的那些节点将被视为同一社区的一部分。

社区发现算法： Louvain 算法-模块度

Modularity函数最初被用于衡量社区发现算法结果的质量，它能够刻画发现的社区的紧密程度。那么既然能刻画社区的紧密程度，也就能够被用来当作一个优化函数，即如果能够提升当前社区结构的modularity，则将结点加入它的某个邻居所在的社区中。

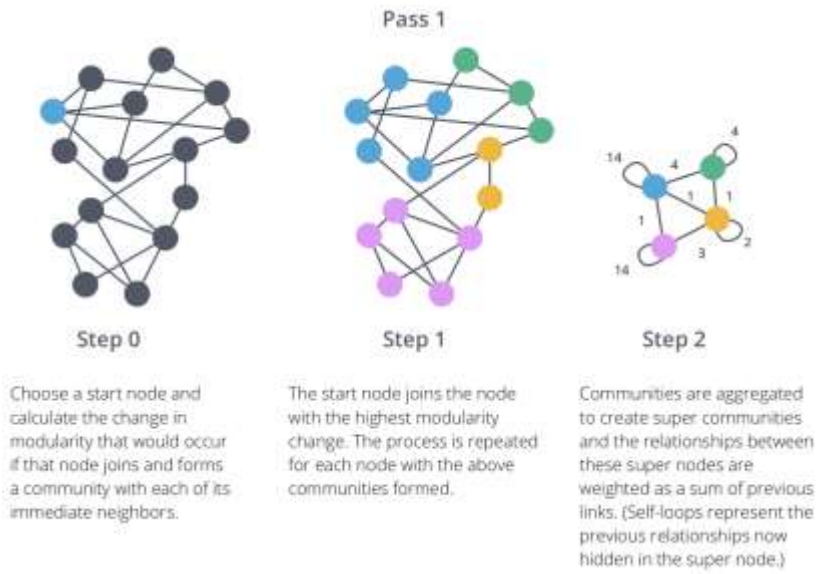
模块度函数

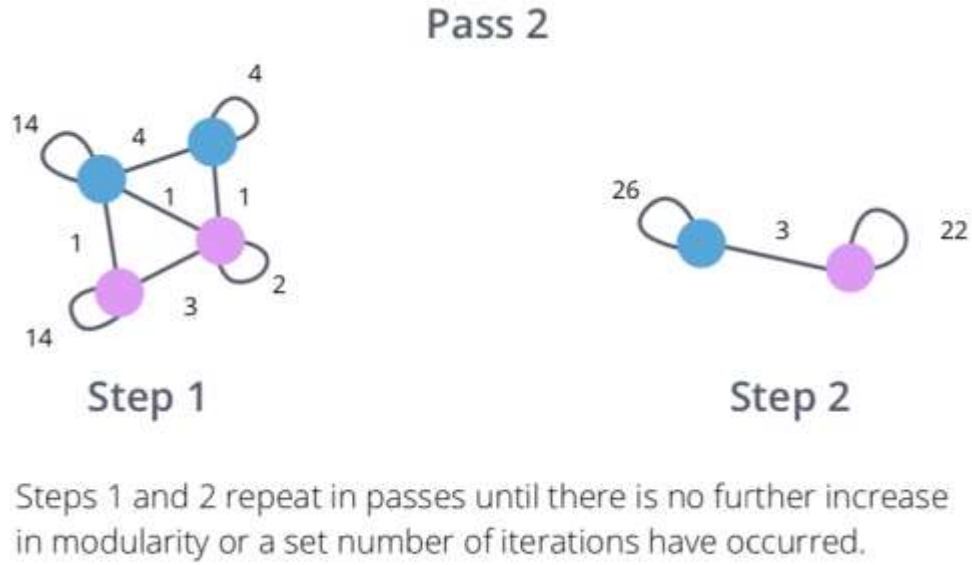
$$Q = \frac{1}{2m} * \sum_{ij} \left[A_{i,j} - \frac{k_i * k_j}{2m} \right] * \delta(C_i, C_j)$$

其中m为图中边的总数量，k_i表示所有指向节点i的连边权重之和，k_j同理。
A_{i,j} 表示节点i，j之间的连边权重。

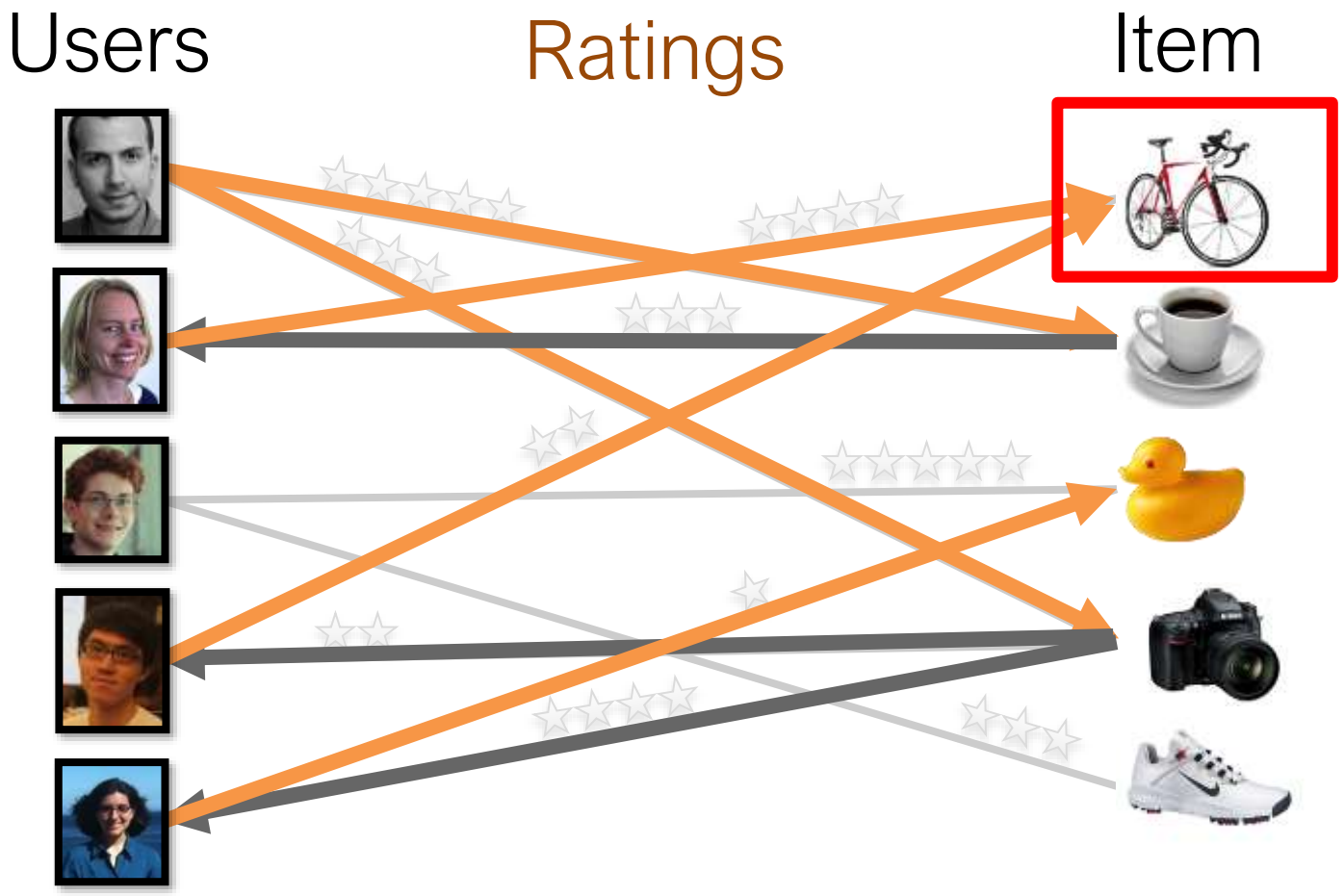
社区发现算法：Louvain 算法

- 核心算法分两步：
  - 算法扫描数据中的所有节点，针对每个节点遍历该节点的所有邻居节点，衡量把该节点加入其邻居节点所在的社区所带来的模块度的收益。并选择对应最大收益的邻居节点，加入其所在的社区。这一过程化重复进行直到每一个节点的社区归属都不在发生变化。
  - 对步骤1中形成的社区进行折叠，把每个社区折叠成一个单点，分别计算这些新生成的“社区点”之间的连边权重，以及社区内的所有点之间的连边权重之和。用于下一轮的步骤1。



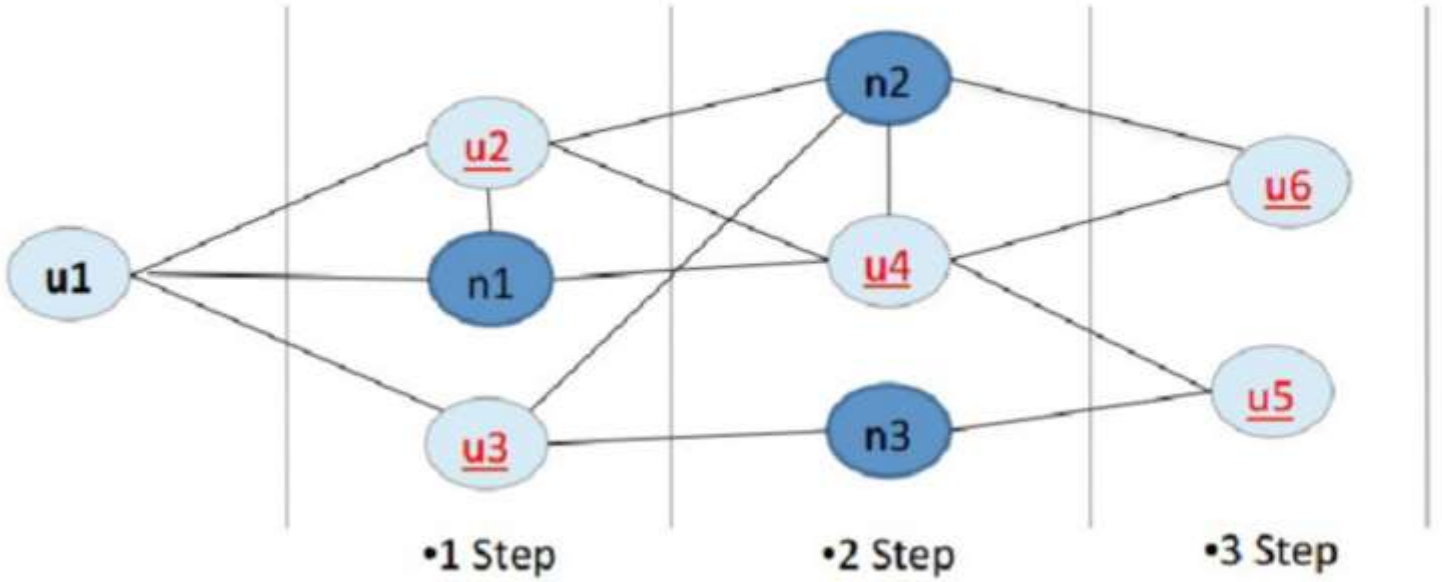


图计算举例： Recommending Products



典型应用：潜在用户发现

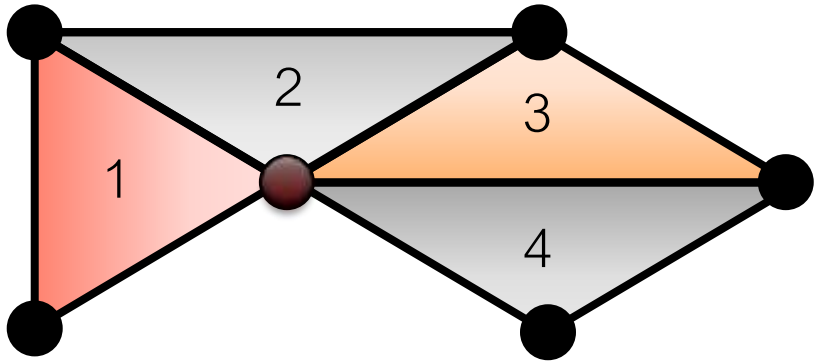
可以通过2步或3步邻居分析找到一个潜在的新用户。



u1-u6是现存用户，n1-n3 是潜在用户。

图计算举例： Finding Communities

- Count triangles passing through each vertex:



- Measures “cohesiveness” of local community

Fewer Triangles
Weaker Community

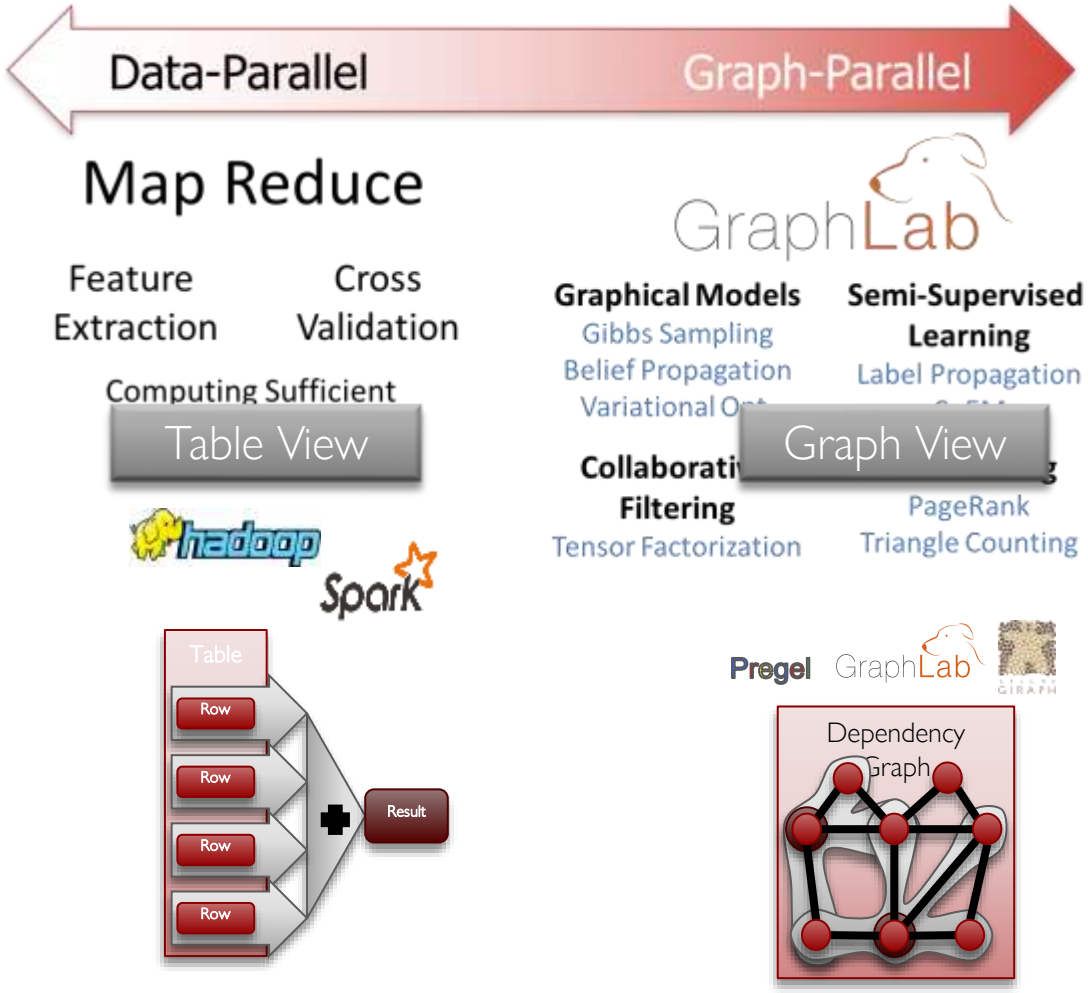
More Triangles
Stronger Community

图计算举例：欺诈检测


欺诈检测
•转账记录组成图结构；
•实时检测资金转账异常行为。
Fraud
https://blog.csdn.net/u0124150

二、图数据并行处理

图计算：从数据并行到图数据并行

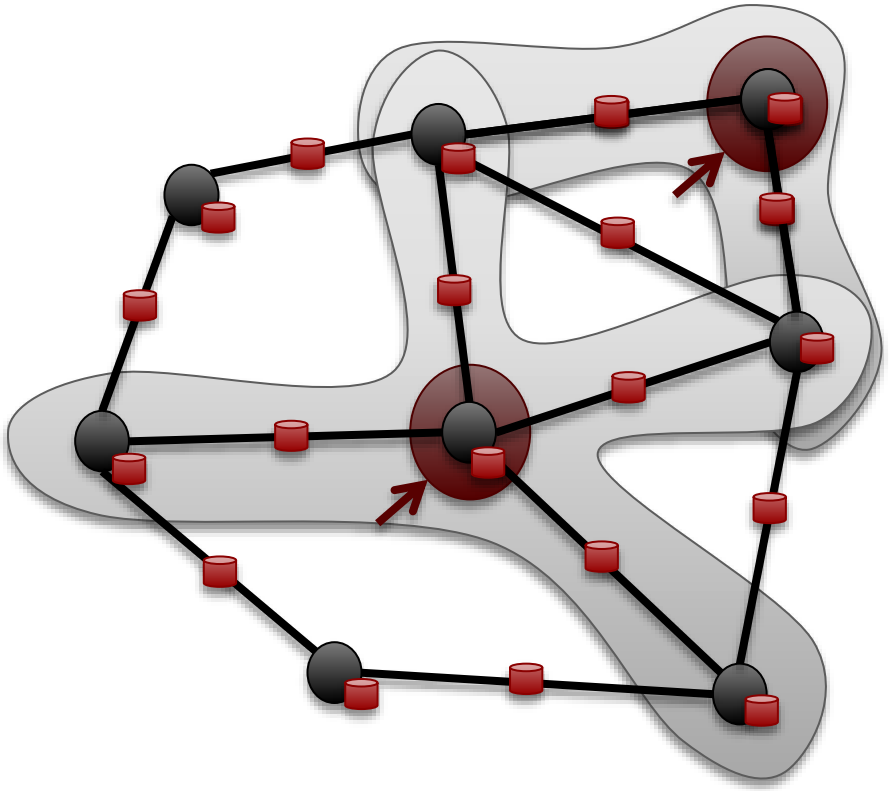


- 简单易用的上层接口，将底层的存储访问、网络通讯、横向扩展、单点容错等相对来说更复杂的问题透明化，达到普通的软件开发人员只需要聚焦于上层应用的目的。
- 针对数据并行计算、图计算、流计算、和矩阵计算等不同类别的应用应该分别使用不同的架构与之对应。

图并行： The Graph-Parallel Pattern

Model / Alg. State

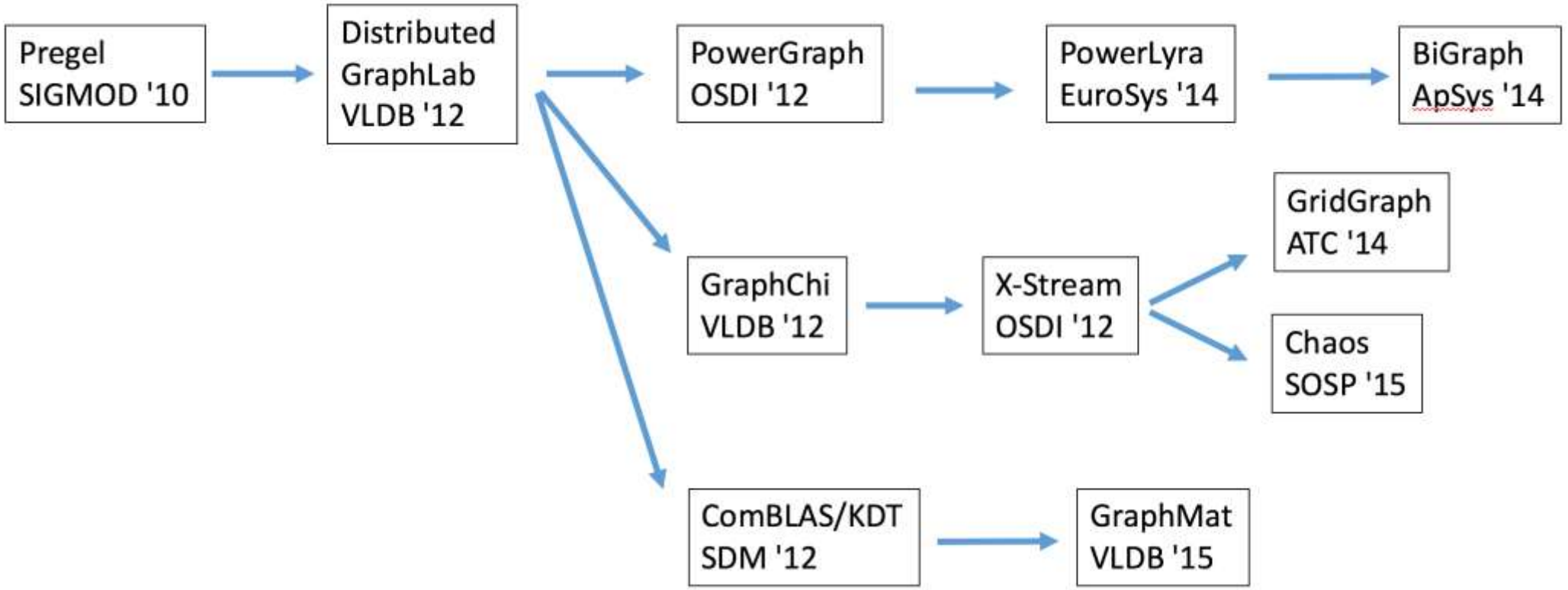
Computation depends only on the **neighbors**



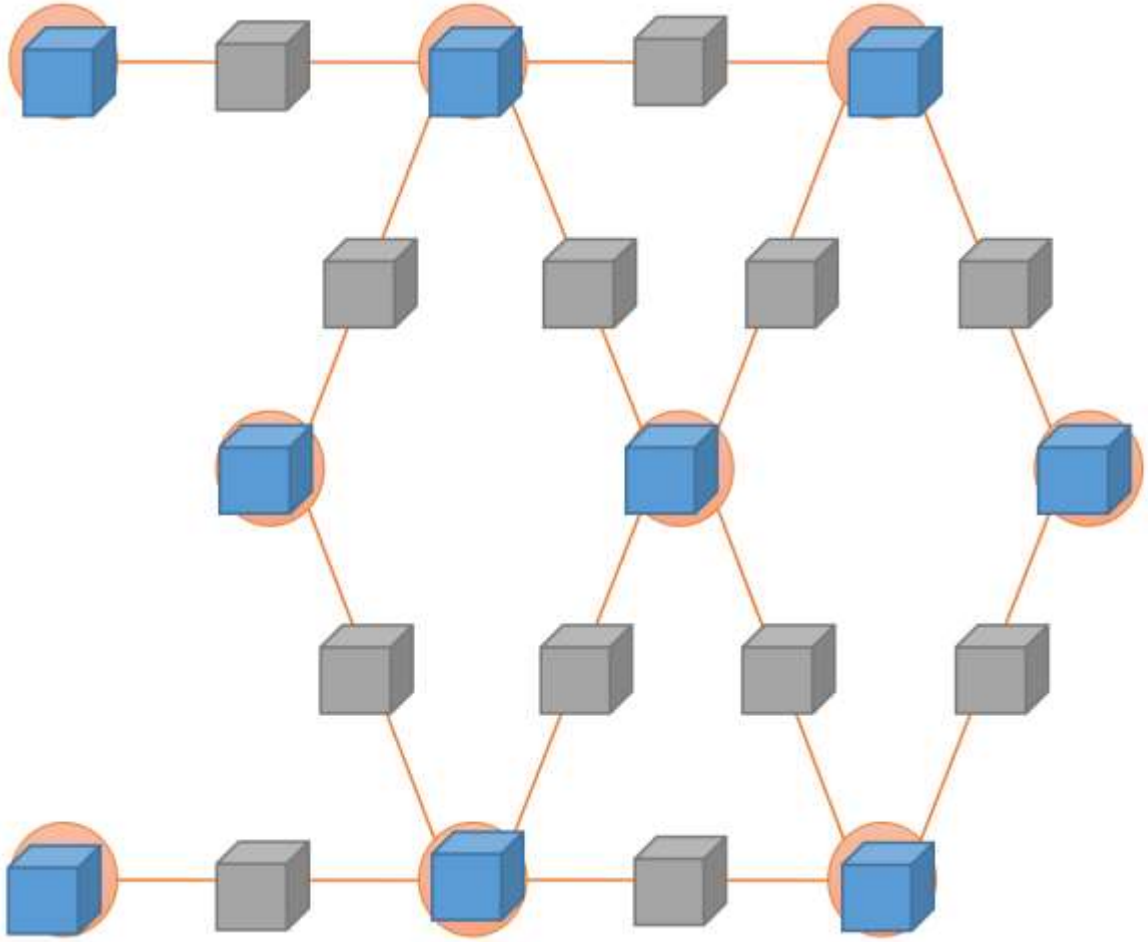
Many Graph-Parallel Algorithms

- Collaborative Filtering
  - Alternating Least Squares
  - Stochastic Gradient Descent
  - Tensor Factorization
- Structured Prediction
  - Loopy Belief Propagation
  - Max-Product Linear Programs
  - Gibbs Sampling
- Semi-supervised ML
  - Graph SSL
  - CoEM
- Community Detection
  - Triangle-Counting
  - K-core Decomposition
  - K-Truss
- Graph Analytics
  - PageRank
  - Personalized PageRank
  - Shortest Path
  - Graph Coloring
- Classification
  - Neural Networks

图并行的发展脉络


Pregel
SIGMOD '10
Distributed
GraphLab
VLDB '12
PowerGraph
OSDI '12
PowerLyra
EuroSys '14
BiGraph
ApSys '14
GraphChi
VLDB '12
X-Stream
OSDI '12
GridGraph
ATC '14
Chaos
SOSP '15
ComBLAS/KDT
SDM '12
GraphMat
VLDB '15

Pregel：Data Graph 和 Vertex-centric Programming



Graph:

- Social Network

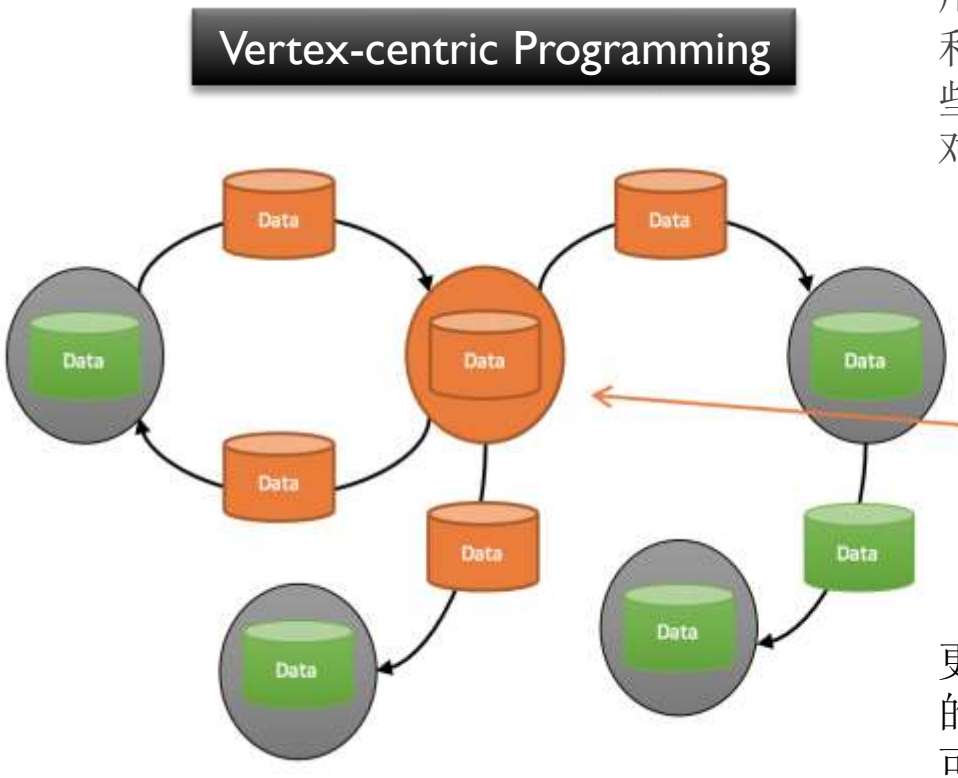
Vertex Property:

- User Profile
- PageRank

Edge Property:

- Distance

以点为中心的编程： Think as a Vertex



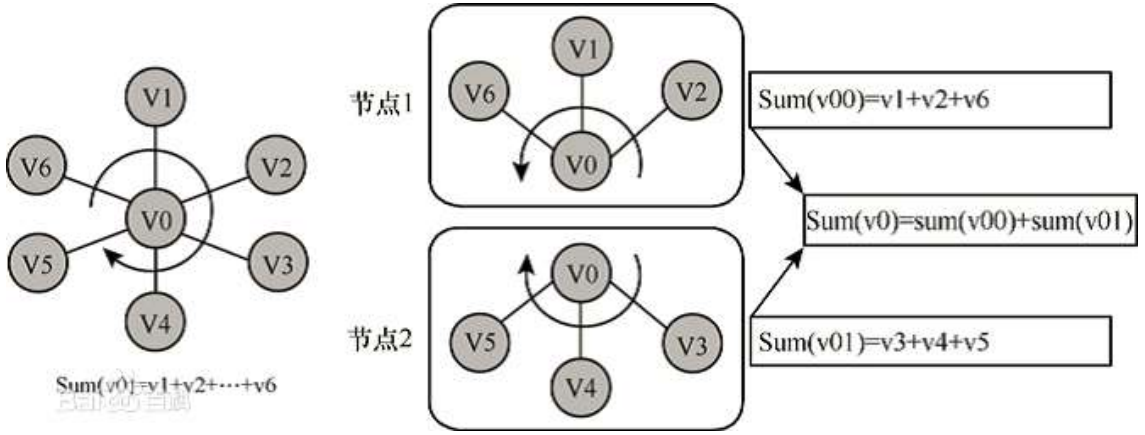
用户所实现的 Vertex Program 的操作范围仅为对应点的临域（即自己的点权和所有相连的边权），而且只被允许读取入边上的值和修改出边上的值。这些限制的主要目的在于方便并行的实现，理论上只要没有操作到共享的数据对应两个点的 Vertex Program 就可以并行地执行。

MyFunc(vertex)
{ // modify neighborhood }

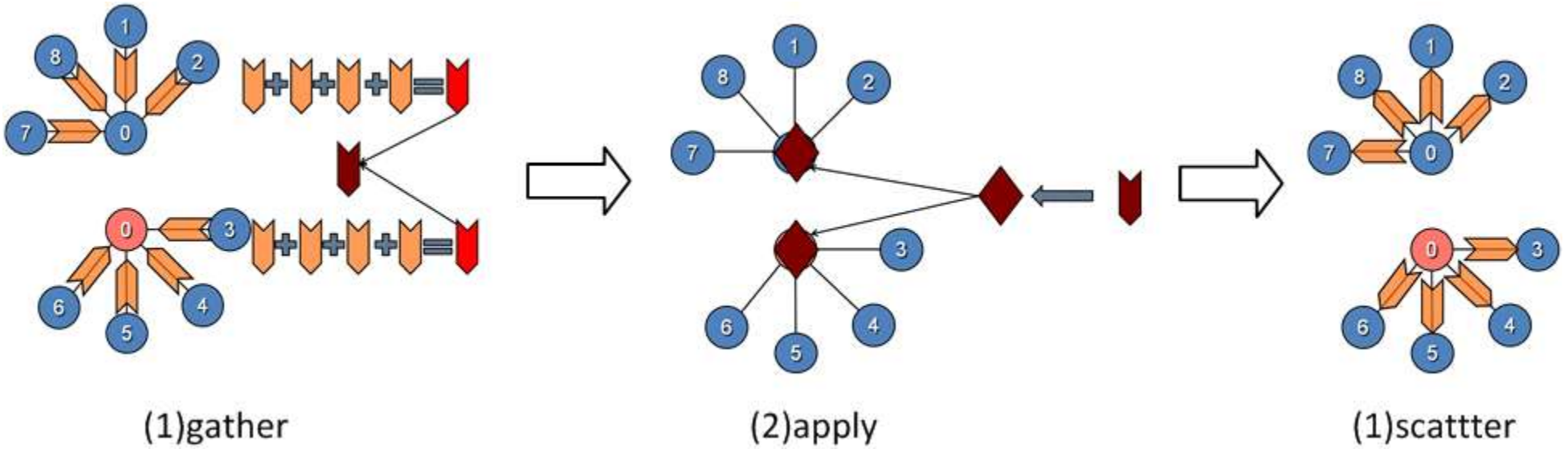
更进一步的，Pregel 的计算系统遵循所谓 Bulk Synchronous Parallell（BSP）的计算模式，即：每一次计算由一系列 superstep 组成，每一个 superstep 又可以分成**本地并发计算、全局通信和同步**三个步骤；在本地计算时不产生任何通讯，全局通讯时也不进行任何计算。每一个 superstep 的本地并发计算阶段就是各个节点分别执行 Vertex Program 的过程，而全局通信阶段则负责把产生的消息送达相应的计算节点。

同步 V.S. 异步

- Pregel 的同步计算模式要求运算速度快的节点每一个 superstep 都必须要等待运行速度慢（或负载更高）的节点，从而造成了大量的浪费。
- 以 GraphLab 为代表的一系列系统都支持被称之为“异步”的计算模式。 在异步模式下不存在 superstep 的概念，每一个计算节点各自维护一个调度队列包含可以被执行的点。
  - 每一个点 Vertex Program 只有当其入边的值被修改了或者被显示地通知了的时候才会被执行；
  - 如果两个点间有边相连，那么它们不会被同时并行地执行；
  - 前者减少了不必要的开销而后者则保证了不会产生数据读写上的冲突。
- 一般来说同步和异步的执行模式各有优劣，异步的模式可以减少负载不均衡带来的性能损失，而负载基本均衡的情况下，同步的模式由于其更小的调度开销速度更快。



GraphLab

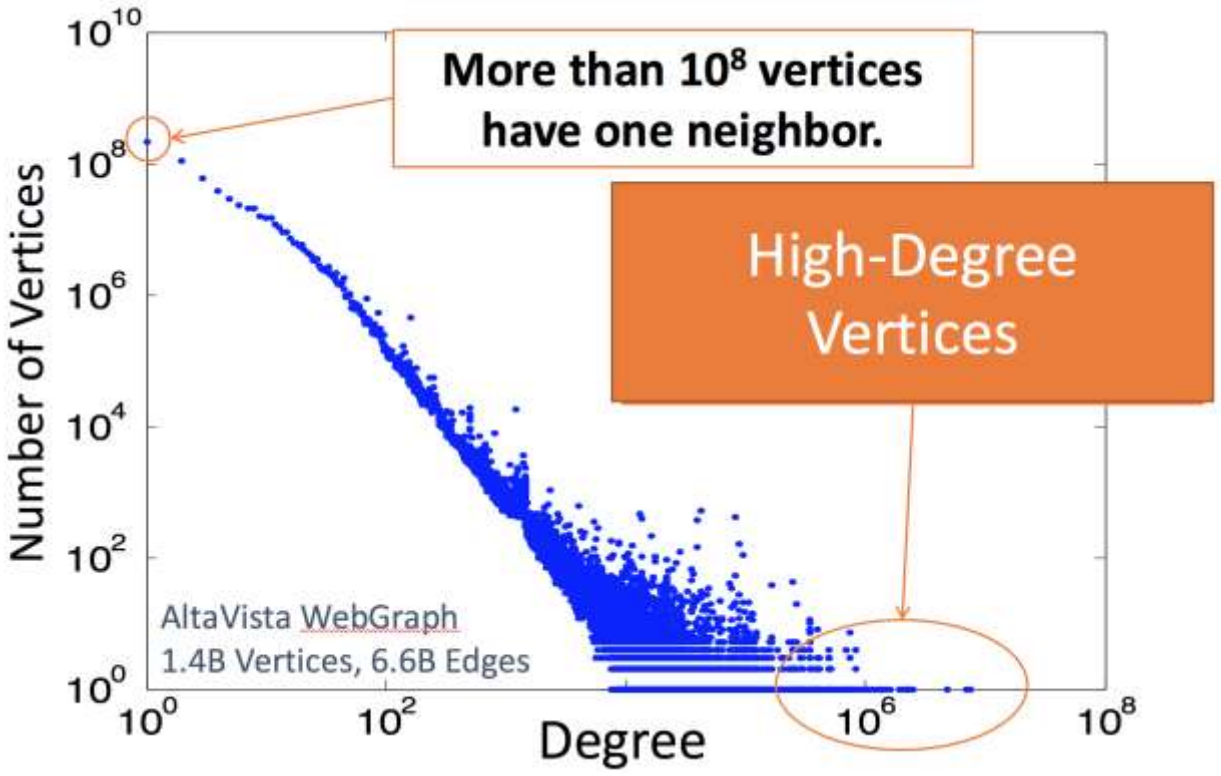


（1）gather 阶段
工作顶点从邻接顶点收集信息，对从邻接点收集的数据被GraphLab 进行求和运算。该阶段所有的顶点和边数据都是只读的。

（2）apply 阶段
各个从节点将gather 节点计算得到的求和值发送到master 节点上，master进行汇总得到总的和，然后Master 再根据业务需求执行一系列计算，更新工作顶点的值。该阶段顶点可修改，边不可修改。

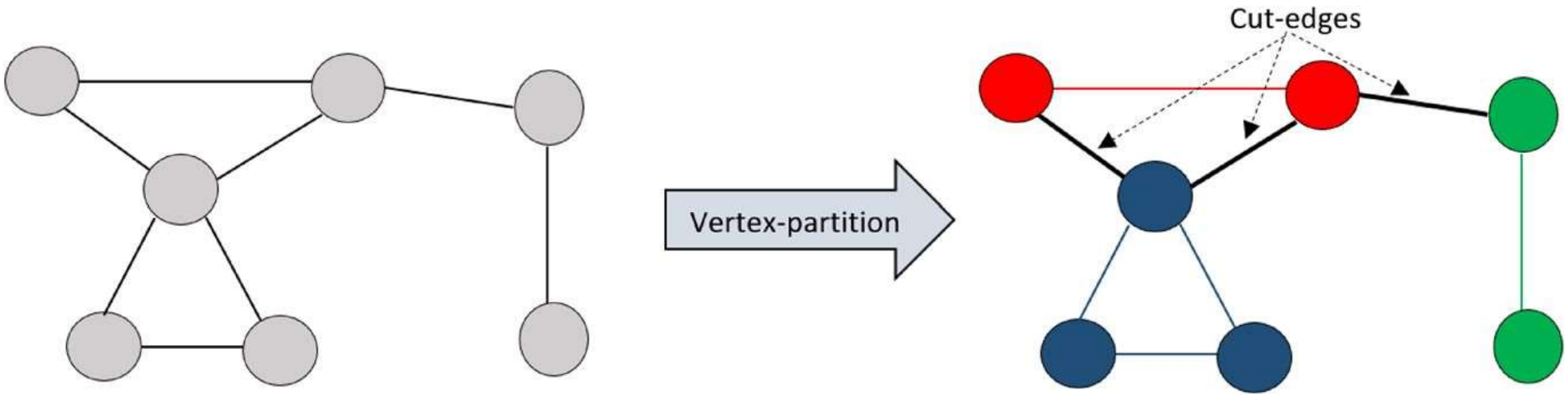
（3）scatter 阶段
工作顶点更新了自己的值后，根据需要可以更新顶点相邻的边信息，并且通知依赖该工作顶点的顶点更新自己的状态。该阶段顶点只读，边数据可写。

图的划分：图的不均衡问题



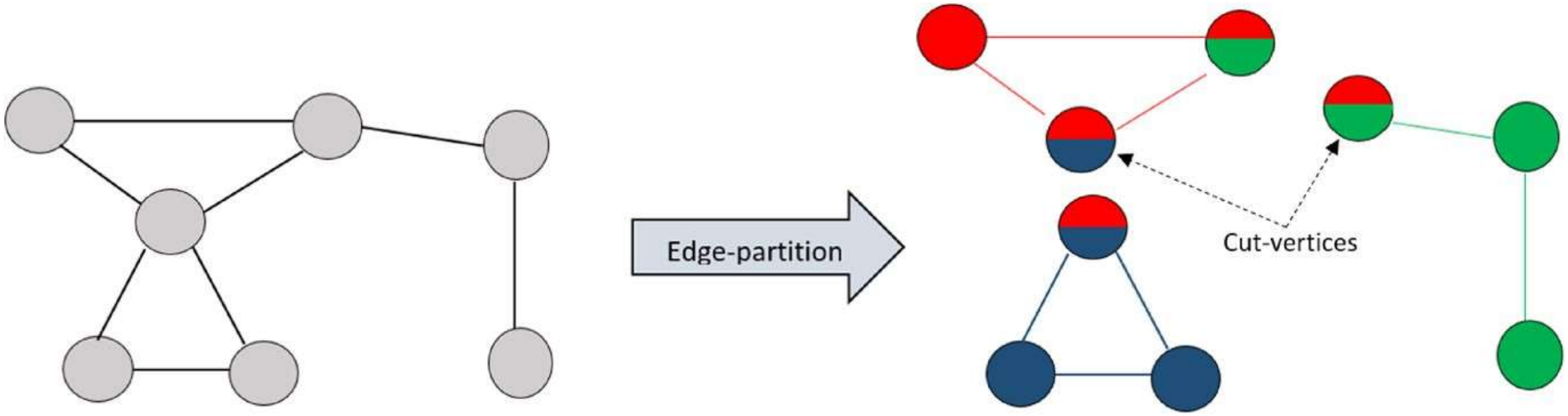
- 早期的 Pregel 和 GraphLab 等系统采用的都是上面提到的基于点的划分方法（Vertex-based），由于被分割的是边，有时也称之为 edge-cut。
- 这种划分在各个点所相邻的边数比较均匀的时候是一个还算可以接受的办法，但问题在于，根据调研实际图计算中常用的图都满足 power law。

顶点划分：切边



顶点划分是将图中的顶点划分为不同的分区，主要的挑战是顶点分割要将切边最小化。

边划分：切顶点

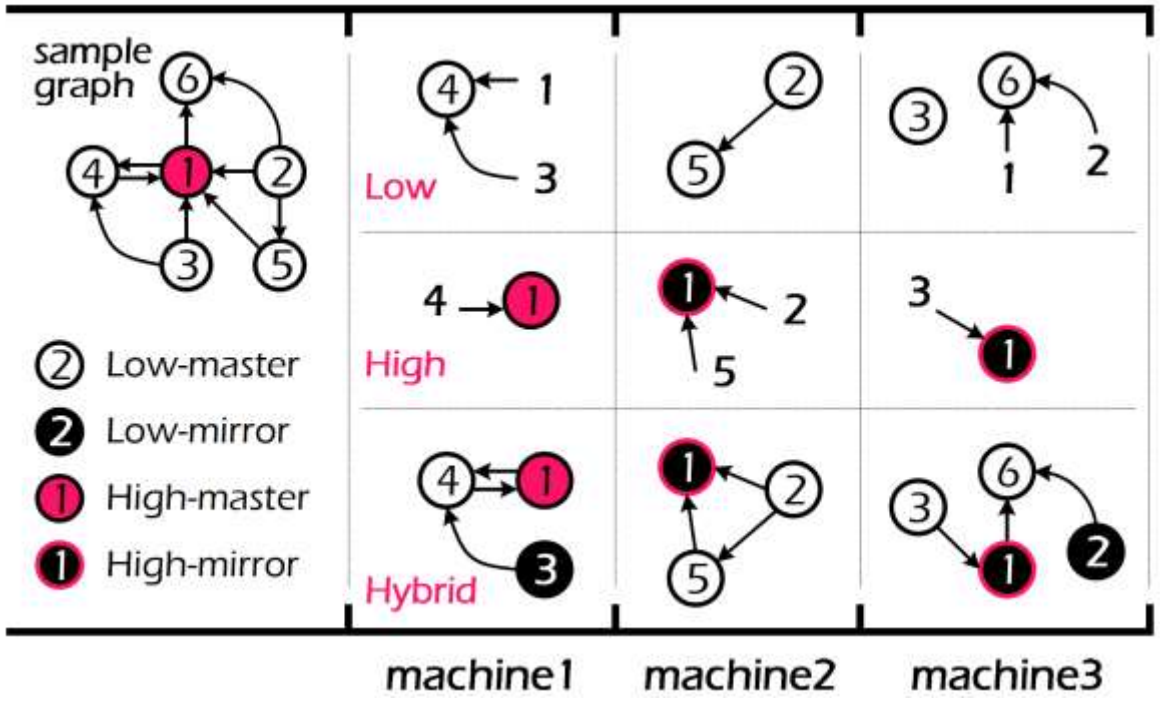


边划分是将每个边都划分到一个确切的分区中，然后拥有不同的分区邻接边的顶点成为切顶点（cut-vertices），边分割面临的问题是最小化切顶点。

如果顶点的度数分布比较平衡，那么顶点分区就更合适。鉴于每个顶点的度是不一样的，枢纽节点的度可能会非常高（幂律度分布），这将导致各个分区边数的不平衡。

通过边分区，我们可以将枢纽顶点划分到不同的分区中去，从而减少切边并且平衡各分区之间的工作量。

混合划分：Hybrid Cut

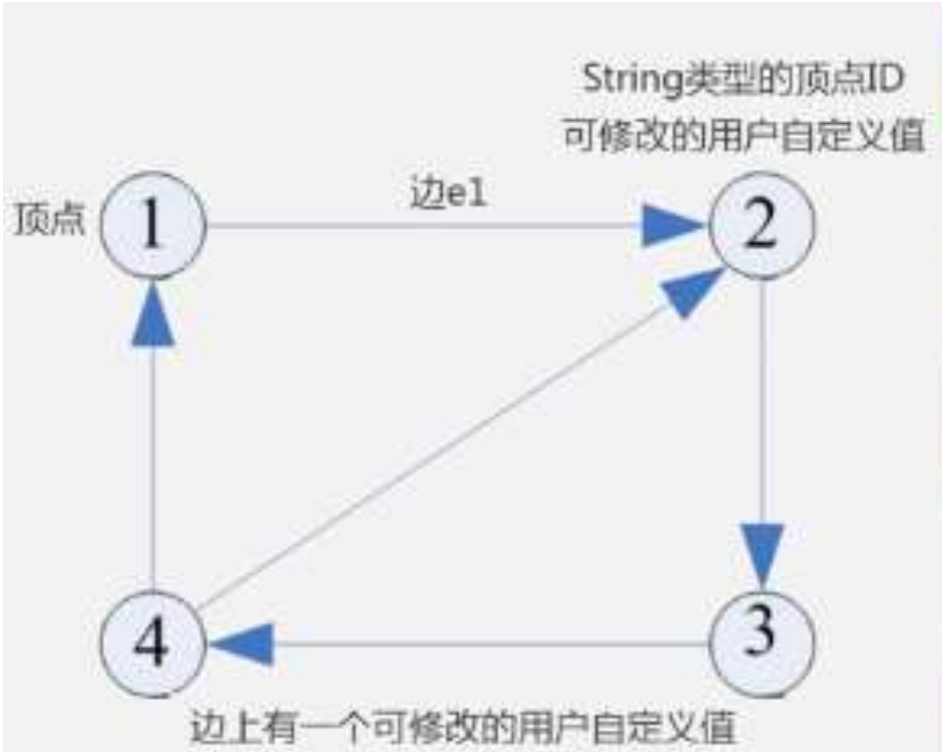


hybrid-cut 可以被看成是 1D 和 2D 划分的混合（对度数小的点用 1D，大的用 2D）

- Hybrid-cut 的方法原理是：当一条边的终点度数小于阈值（图中为 3，一般设为 100～200）时则将这条边按其终点的 hash 分配，反之则按源点的 hash 分配。
  - 图中除 1 以外的所有点入度都小于阈值，所以它们被均匀分配给三个计算节点，并附带其所有入边。
  - 相对的，对于高度数的节点 1，它的 4 条入边被均分给了各个计算节点。
  - 这样做的依据可以概括为: 度数大的点数量稀少且很难保证其 replica 的数量，因此优先保证数量众多的度数小的点不存在 replica。

三、Pregel图计算系统实现原理

Pregel： 基于BSP 模型实现的并行图计算系统



01 Pregel计算模型以有向图作为输入

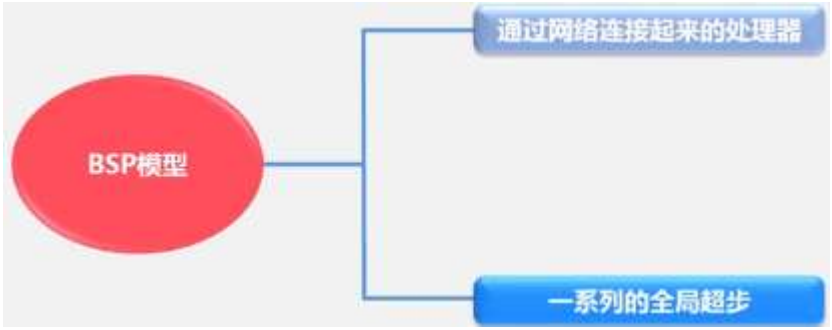
02 有向图的每个顶点都有一个String类型的顶点ID

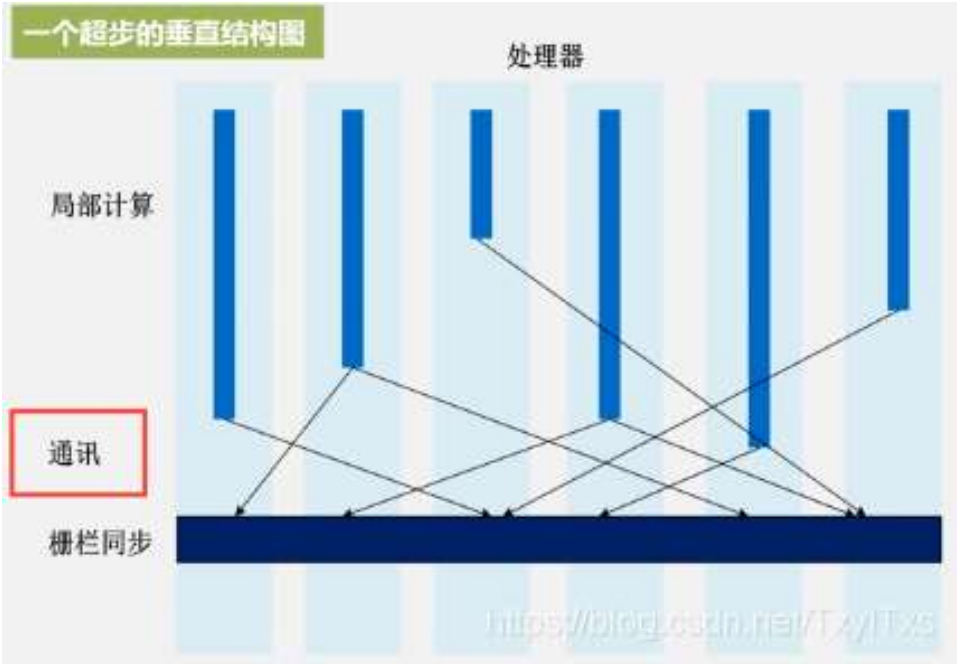
03 每个顶点都有一个可修改的用户自定义值与之关联

04 每条有向边都和其源顶点关联，并记录了其目标顶点ID

05 边上有一个可修改的用户自定义值与之关联

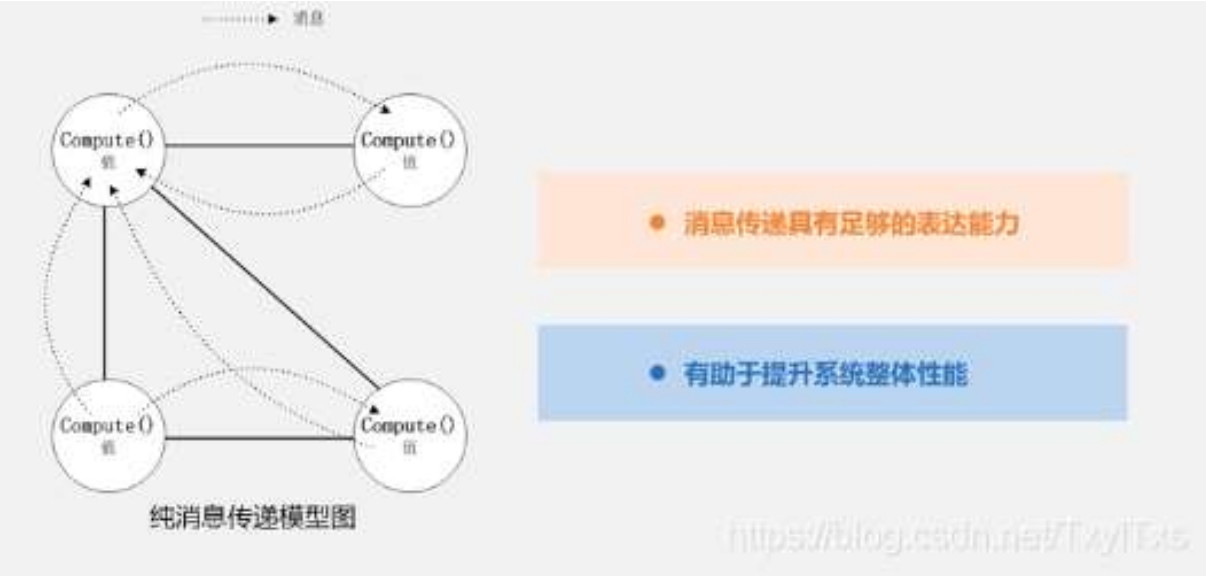
BSP模型： Bulk Synchronous Parallel Computing Mode

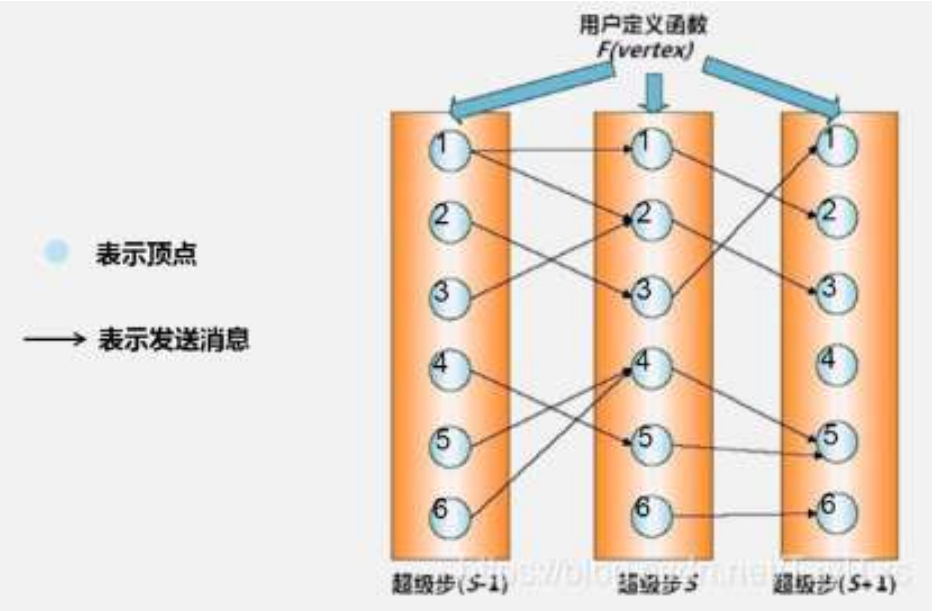




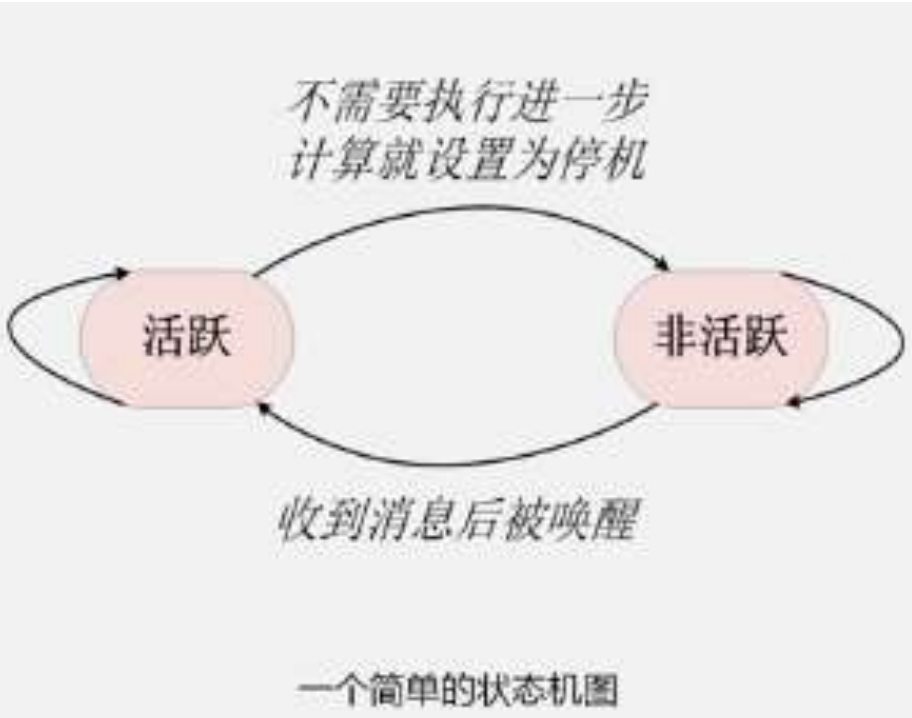
超步： Superstep

消息传递


消息
Compute()
Compute()
Compute()
Compute()
消息传递具有足够的表达能力
有助于提升系统整体性能
纯消息传递模型图


用户定义函数
F(vertex)
表示顶点
表示发送消息
超级步(S-1)
超级步S
超级步(S+1)

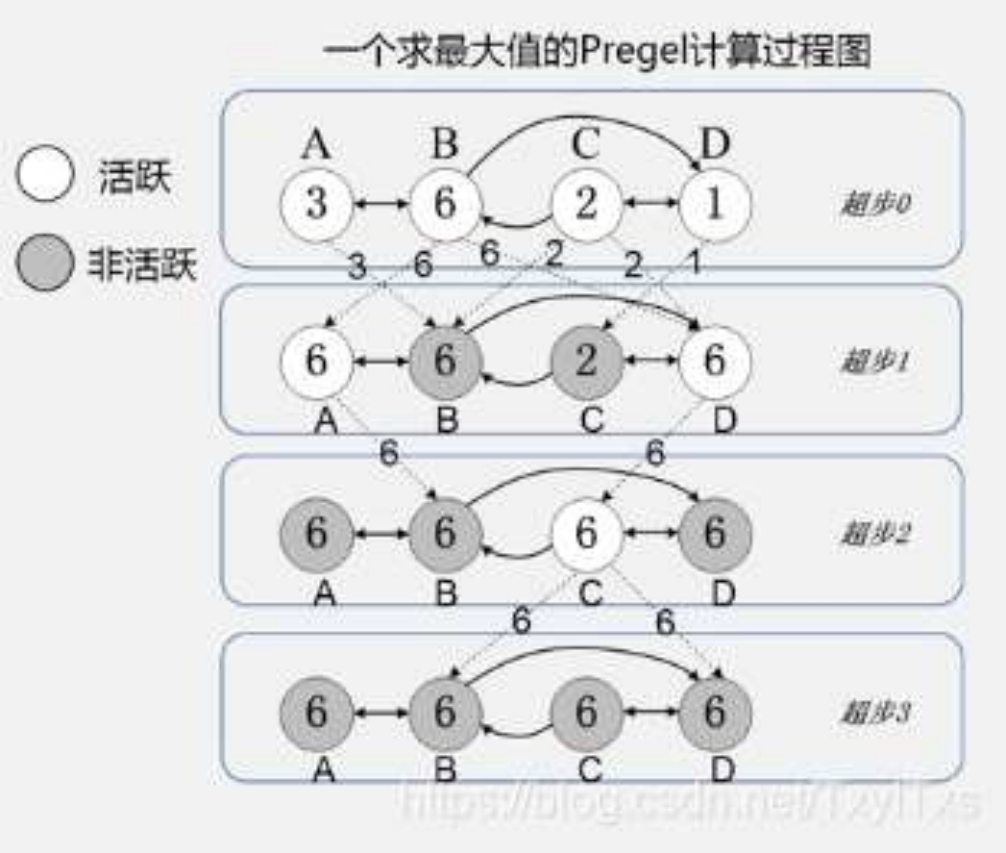
Pregel 的计算过程



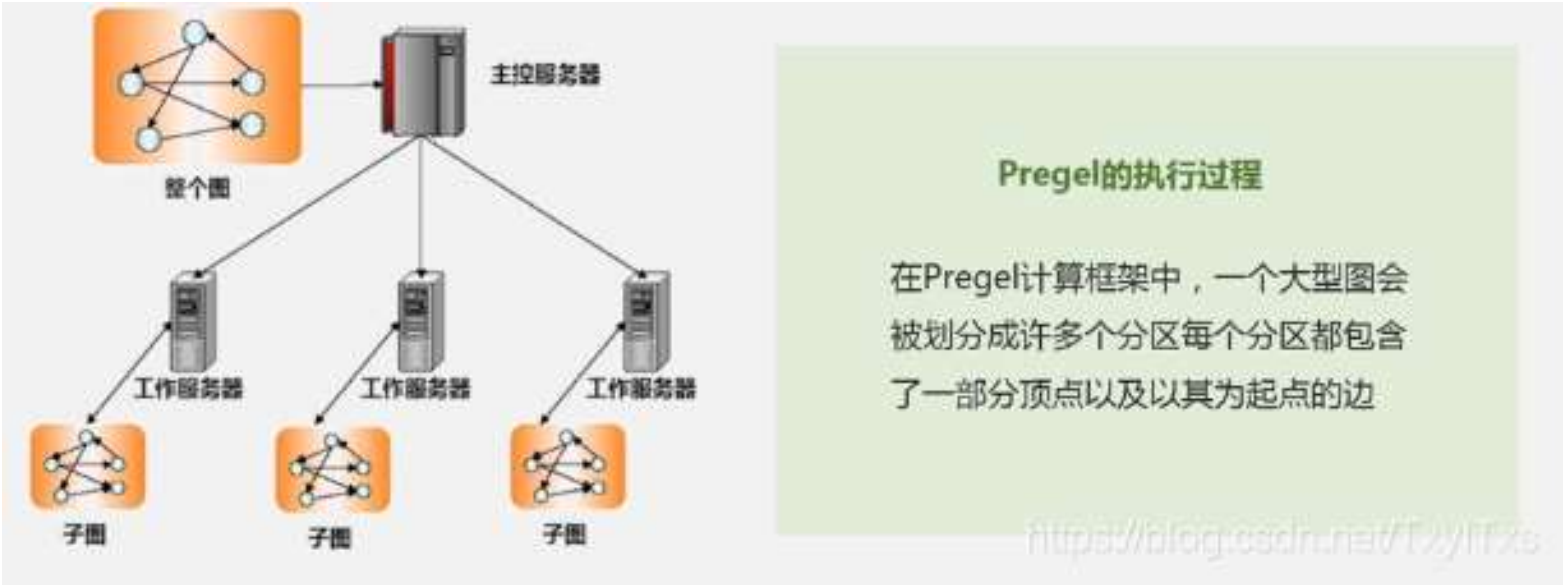
一个简单的状态机图

1. 在Pregel计算过程中，一个算法什么时候结束是由所有顶点的状态决定
2. 在第0个超步，所有顶点处于活跃状态
3. 一个顶点不需要继续执行进一步的计算时就会把自己的状态设置为“停机”
4. Pregel计算框架必须根据条件判断来决定是否将其显式地置进入活跃状态

Pregel 实例



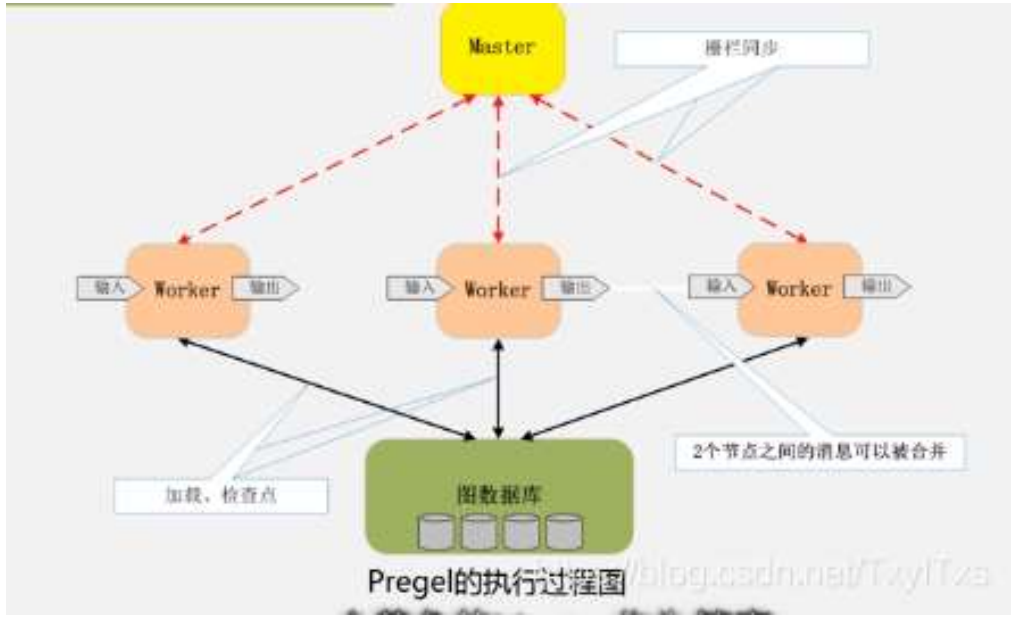
Pregel执行过程



一个顶点应该被分配到哪个分区上

是由一个函数决定的，系统默认函数为hash(ID) mod N
其中，N为所有分区总数，ID是这个顶点的标识符

Pregel执行过程



Pregel的执行过程图

•选择集群中的多台机器执行图计算任务，有一台及其会被选为Master 其他机器作为Worker;
•Master把一个图分成多个分区，并把分区分配到多个Worker,一个Worker会领到一个或多个分区，每个Worker知道所有其他Worker所分配到的分区情况;

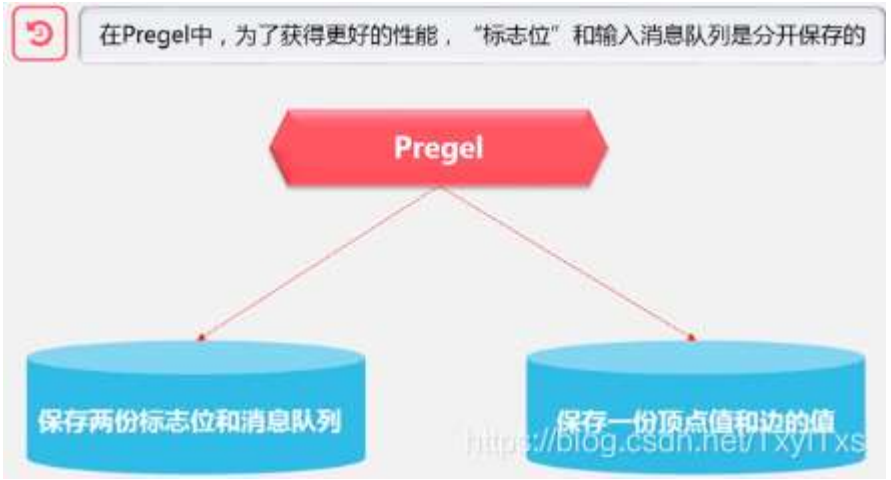
Master会把用户输入划分成多个部分然后，Master会为每个Worker分配用户输入的一部分。如果一个Worker从输入内容中加载到的顶点，刚好是自己所分配到的分区中的顶点，就会立即更新相应的数据结构

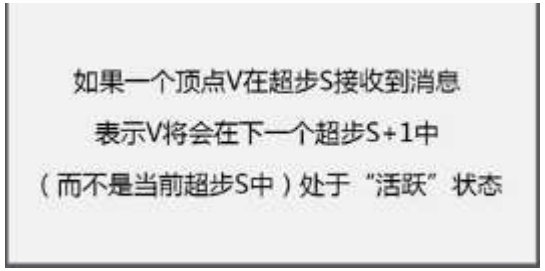
否则，该Worker会根据加载到的顶点的ID，把它发送到其所属的分区所在的Worker上。当所有的输入都被加载后，图中的所有顶点都会被标记为“活跃”状态

Master 向每个Worker发送指令，Worker收到指令后，开始运行一个超步。当一个超步中的所有工作都完成以后，Worker会通知Master，并把自己在下一个超步还处于“活跃”状态的顶点的数量报告给Master

计算过程结束后，Master会给所有的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

Worker





- 一般在执行过程中Worker的信息保存在内存当中，包括：顶点当前的值，出射边列表，消息队列，标志位；
- worker会对自己所管辖的分区中的每个顶点进行遍历，并调用顶点上的Compute() 函数，Computer 函数接收顶点当前值，消息迭代器和出射边迭代器三个参数。

Master

Master指令

Master向所有处于有效状态的Worker发送相同的指令，然后等待Worker回应，在指定时间内某一个Worker没有回应说明这个Worker已经失效了，Master会进入恢复的模式

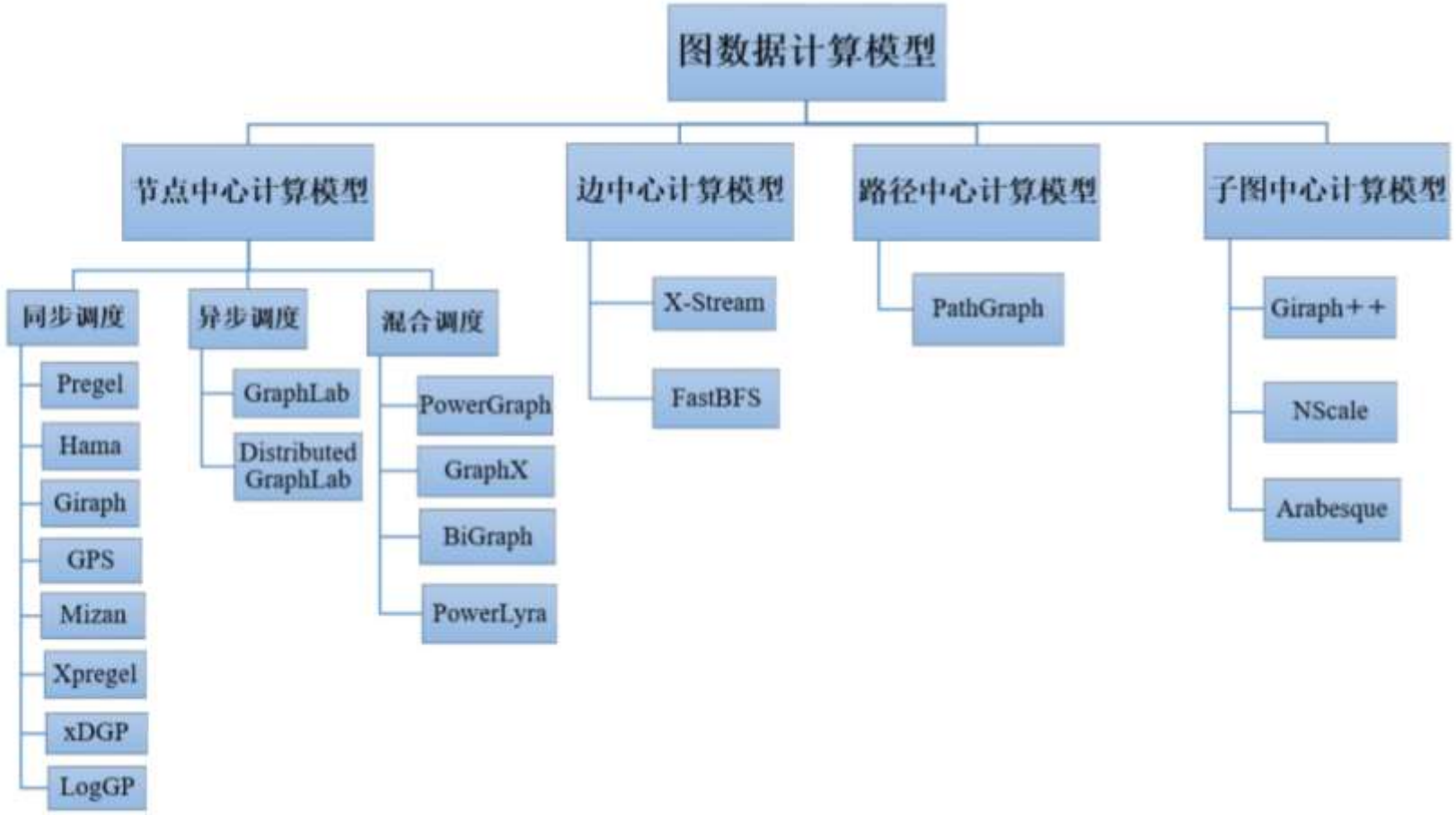
Aggregator

- 每个用户自定义的Aggregator都会采用聚合函数对一个值集合进行聚合计算得到一个全局值。
- 每个Worker都保存了一个Aggregator的实例集，其中的每个实例都是由类型名称和实例名称来标识的。
- 在执行图计算过程的某个超步S中，每个Worker会利用一个Aggregator对当前本地分区中包含的所有顶点的值进行归约，得到一个本地的局部归约值。
- 在超步S结束时，所有Worker会将所有包含局部归约值的Aggregator的值进行最后的汇总，得到全局值，然后提交给Master。
- 在下一个超步S+1开始时，Master就会将Aggregator的全局值发送给每个Worker。

容错性

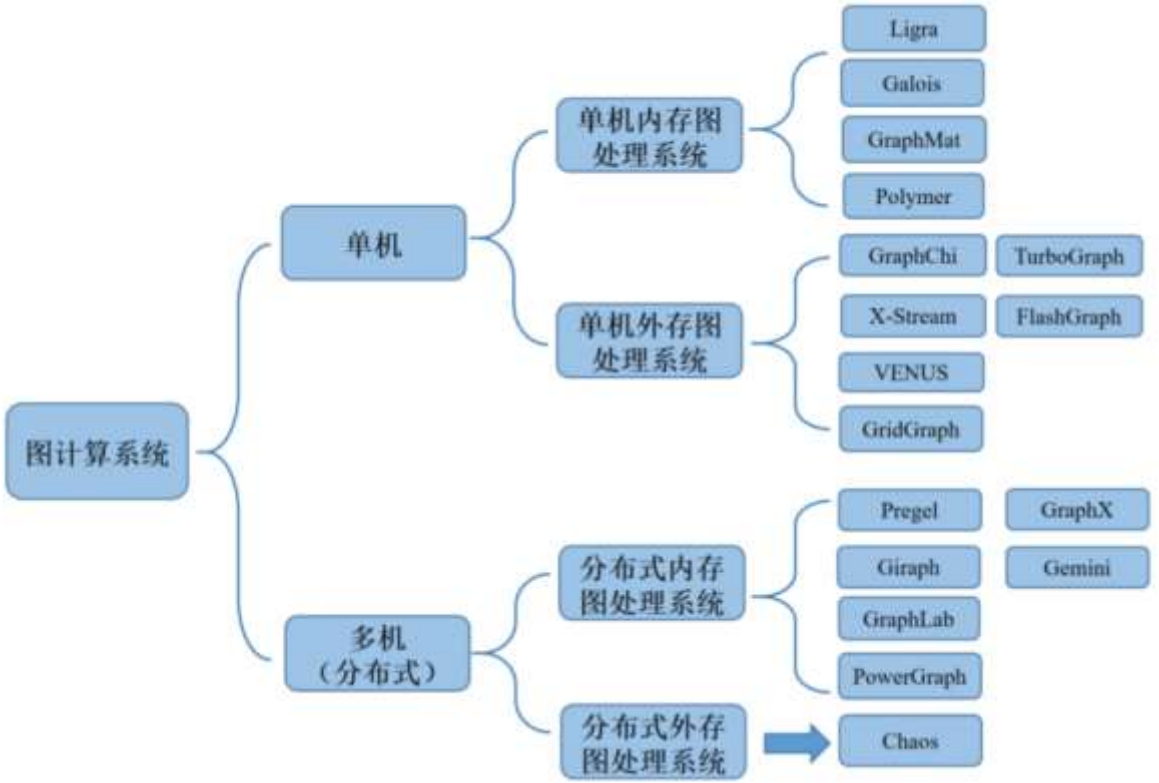

容错性
当一个Worker发生故障时，它所负责维护的分区的当前状态信息就会丢失
Master监测到一个Worker发生故障“失效”后会把失效Worker所分配到的分区，重新分配到其他处于正常工作状态的Worker集合上
Pregel采用检查点机制来实现容错。在每个超步的开始，Master会通知所有的Worker把自己管辖的分区的状态写入到持久化存储设备
Master会周期性地向每个Worker发送ping消息，Worker收到ping消息后会给Master发送反馈消息，每个Worker上都保存了一个或多个分区的状态信息

总结：图计算模型



AMiner《图计算研究报告》

总结：图计算系统



AMiner《图计算研究报告》

谢谢大家！