Group Reassignment for Dynamic Edge Partitioning

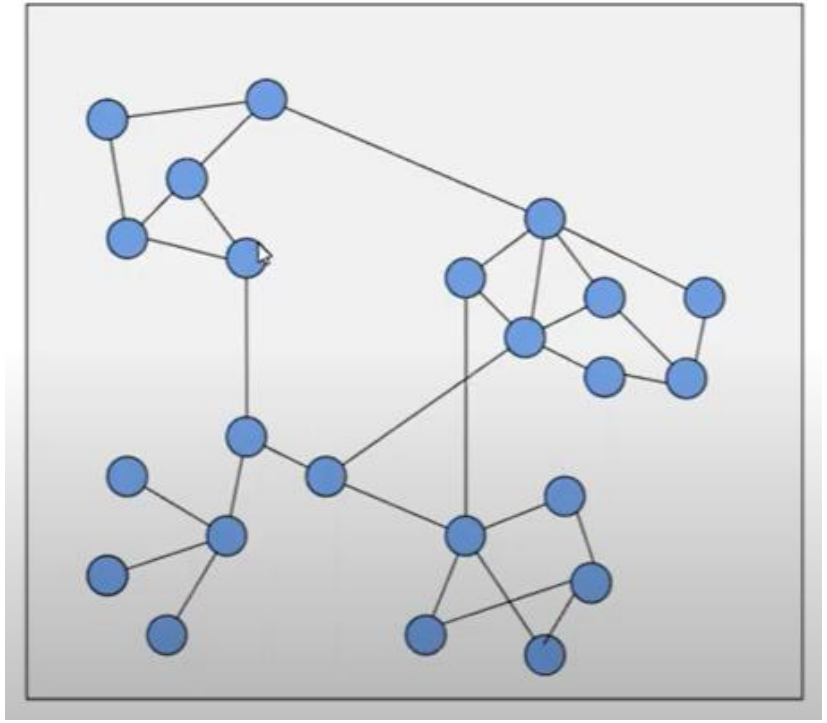
王源博M202173707

background

许多大数据应用都作用于图上，不仅有最短路径，网页排序等传统应用，还包括社交网络分析、生物信息网络分析、道路交通管理等新兴应用。随着图数据量的增长，图的存储和处理要在分布式环境中执行。为了满足这一需求人们提出了许多分布式图处理系统。

background

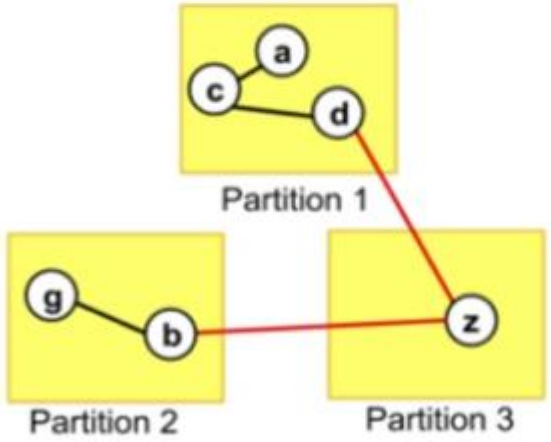
图划分是分布式图处理的基础，它将一个图划分成多个部分。划分的好坏对分布式图处理的性能有很大的影响。
首先处理速度取决于最慢的部分，因此要划分平衡。
其次在分布式图应用中，一些顶点需要与远端的邻居通信来传递消息，这导致了部件间(inter-part)通信。由于网络延迟，部件间通信比部件内通信成本高。
因此，基于分区负载均衡的图划分的目标是最小化子图间通信开销。然而，平衡图划分是一个NP难题。



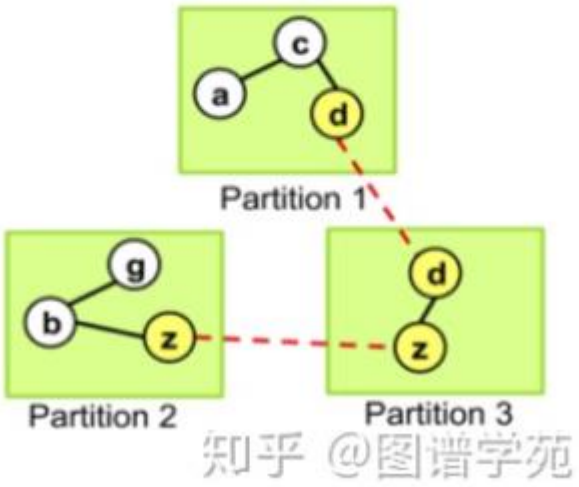
background

图的划分经历了从顶点划分到边划分的转变。
已经从理论上证明了边划分在自然幂律图的分布式图计算中的优越性。
在边划分中，每条边被分配到一个部分，一个顶点的相邻边可能不在同部分，因此需要在对应的部分创建顶点副本，避免了由于超负荷引起的负载不平衡。
因为沿边进行计算，一个划分部分（part）的载荷由边数决定。
为了降低副本间同步的子图间（inter-part）通信开销，边划分需要在分区负载平衡的基础上最小化顶点副本（点割）。

- 点划分（边割）：



- 边划分（点割）：

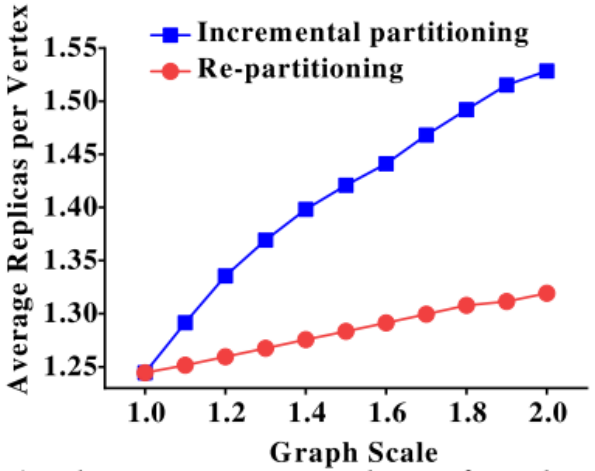


challenge

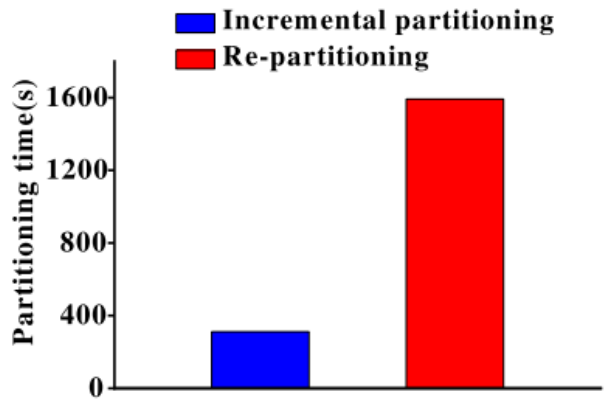
现有的边划分算法大都是静态的，从头划分图，没有考虑图的动态变化。然而现实世界的图都是不断变化的。
对于现实世界的图的研究表明，现实世界的图的在动态的变化过程中稠密度会不断增大，而直径会不断减小。
在初始划分的基础上，图的动态变化会子图间的连接更加紧密，使子图间的通信开销变大。

challenge

现有的动态划分算法大致分为两类：增量划分和重划分。
重划分算法是指每隔一段时间就对整个图重新划分。虽然可以保证副本冗余较小，
但是划分时间较长。



(a) The average number of replicas per vertex at different graph sizes.



(b) The comparison of partitioning time.

Fig. 1. The comparison of applying an incremental and a re-partitioning algorithm on a dynamic graph.

GR-DEP 动态边划分算法——motivation

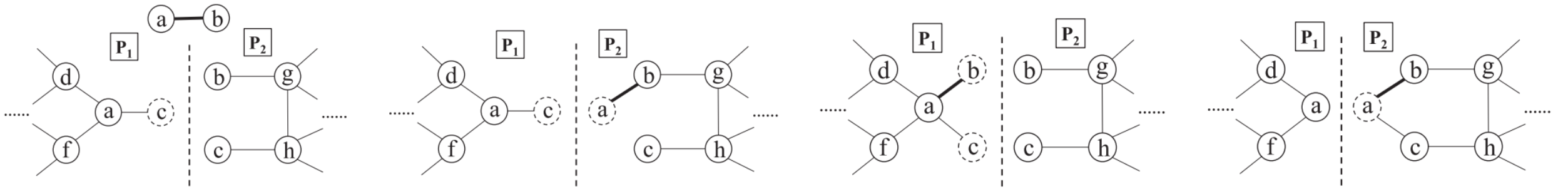
作者发现插入和删除操作都会导致局部次优划分，从而导致冗余副本。
为了消除局部次优划分，作者将联系紧密的边视为一个组（group），通过重新分配另一个与当前部分连接更加紧密的部分的组来减少局部的边割（edge cut）。
在GR—DEP中通过一个分布流对每一个输入的动态变化进行划分通过局部重分配的方法实时的提高划分质量。

GR-DEP 动态边划分算法

- 提出了一种在线处理图结构变化的动态边划分算法。通过对少量边进行重分配，可以实时提高划分质量。
- 讨论了不同的动态变化对划分质量的影响，发现局部次优划分是不可避免的，这是动态图中存在大量冗余边割副本的主要原因。
- 证明了将一些联系紧密的边作为一个group迁移可以改善局部次优划分，我们设计了一种结构感知的方法和一种以自我为中心的搜索并迁移group的方法。将它们应用于不同的顶点，可以取得较好的效果。

GR-DEP 动态边划分算法——问题抽象

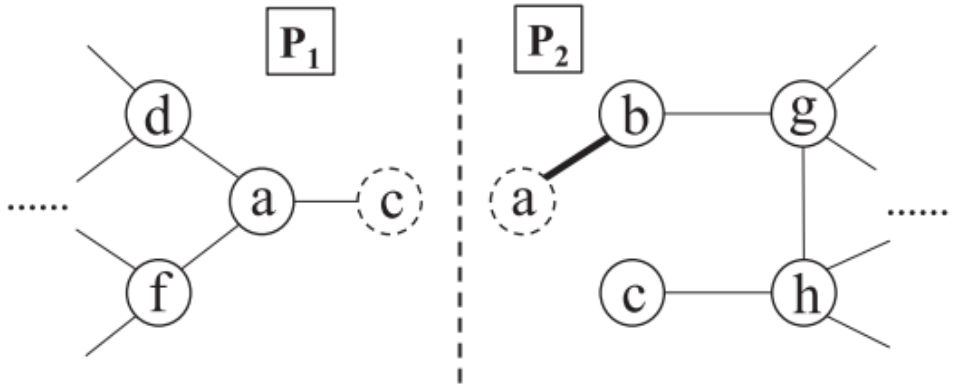
局部次优划分与局部最优划分



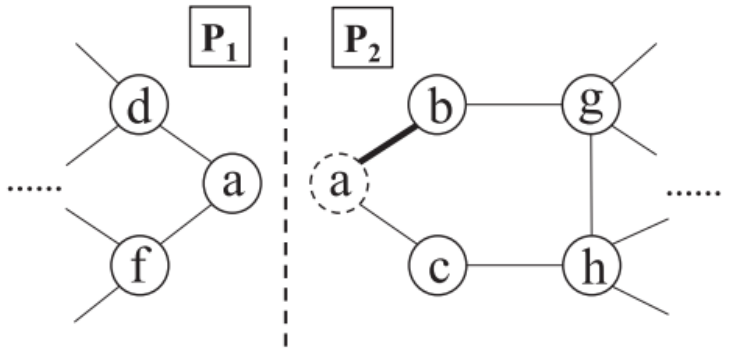
(a) The local initial partitioning. (b) Assigning (a, b) into P_2. (c) Assigning (a, b) into P_1. (d) The local optimal partitioning.

GR-DEP 动态边划分算法——边组重分配

边组的定义——Group of Edge
在下图中，边组是P1中一组连接的边，如果把他迁移到P2中，可以减少划分的割点数。
P1中边={(a,b),(a,c),(a,d),(a,f)}
P1划分为S1={(a,d),(a,f)},S2={(a,b)(a,c)}
把s2迁移到P2中可以减少点割数，s2就是一个边组(edge group)
所以迁移EG可以减少局部的点割



(b) Assigning (a, b) into P_2.

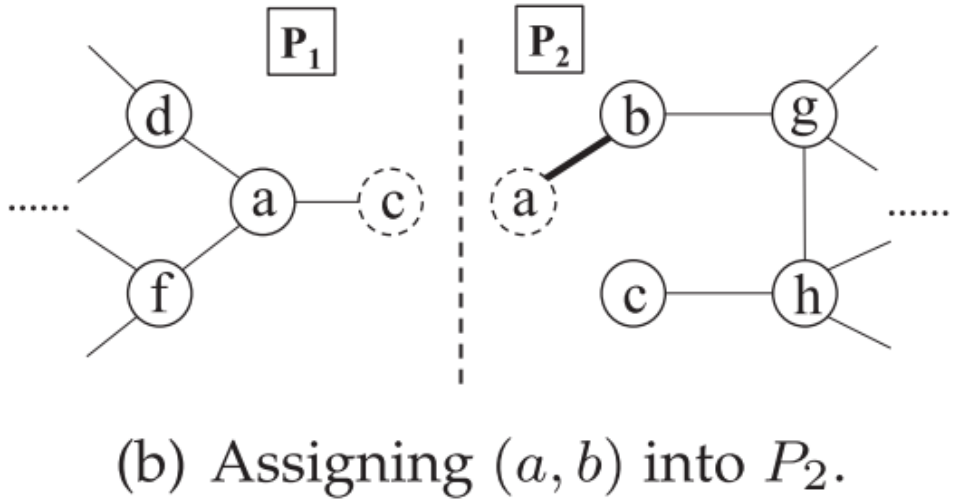


(d) The local optimal partitioning.

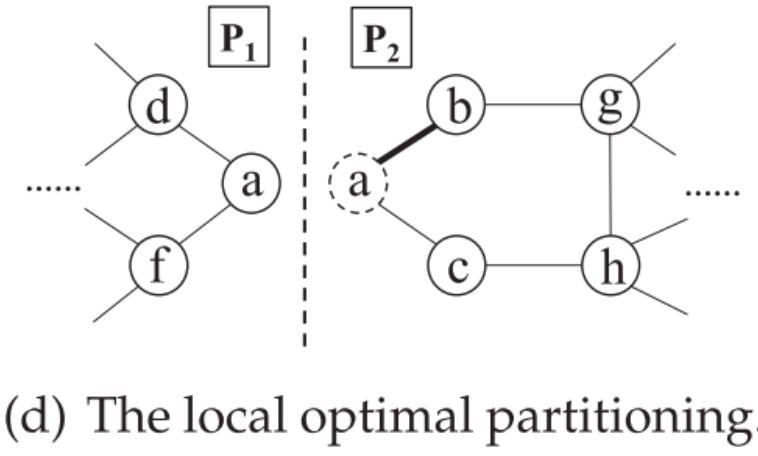
GR-DEP 动态边划分算法——边组重分配

定义：
外部点割（External Vertex-Cut）：$V(EG) \cap VC(S_i, P_j)$ 在图b中evc={a, b, c}
内部点割（Internal Vertex-Cut）：$V(EG) \cap V(P_i \setminus EG)$ 在图b中ivc={a}

evc是EG中当前切割的点，ivc是迁移后需要切割的点
迁移的获得可以定义为 $gain_{i \to j} = |evc| - |ivc|,$



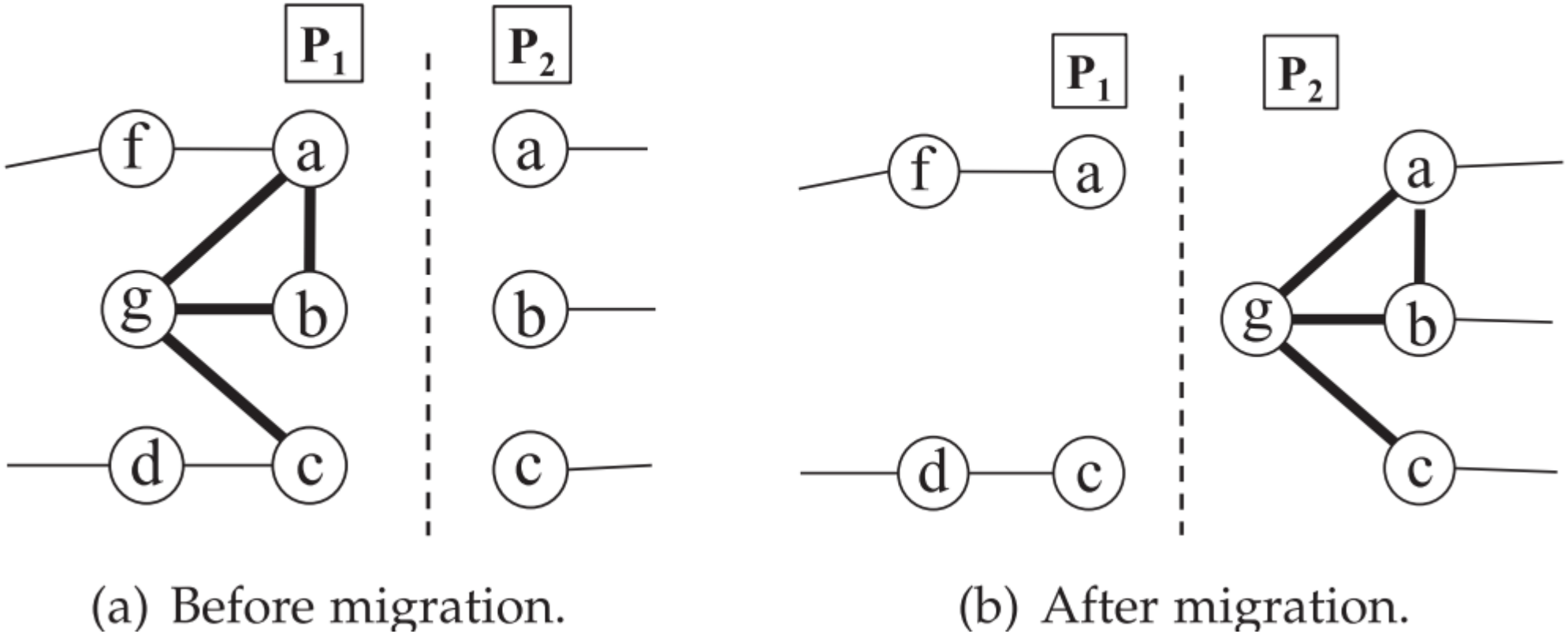
(b) Assigning (a, b) into P_2.



(d) The local optimal partitioning.

GR-DEP 动态边划分算法——边组重分配

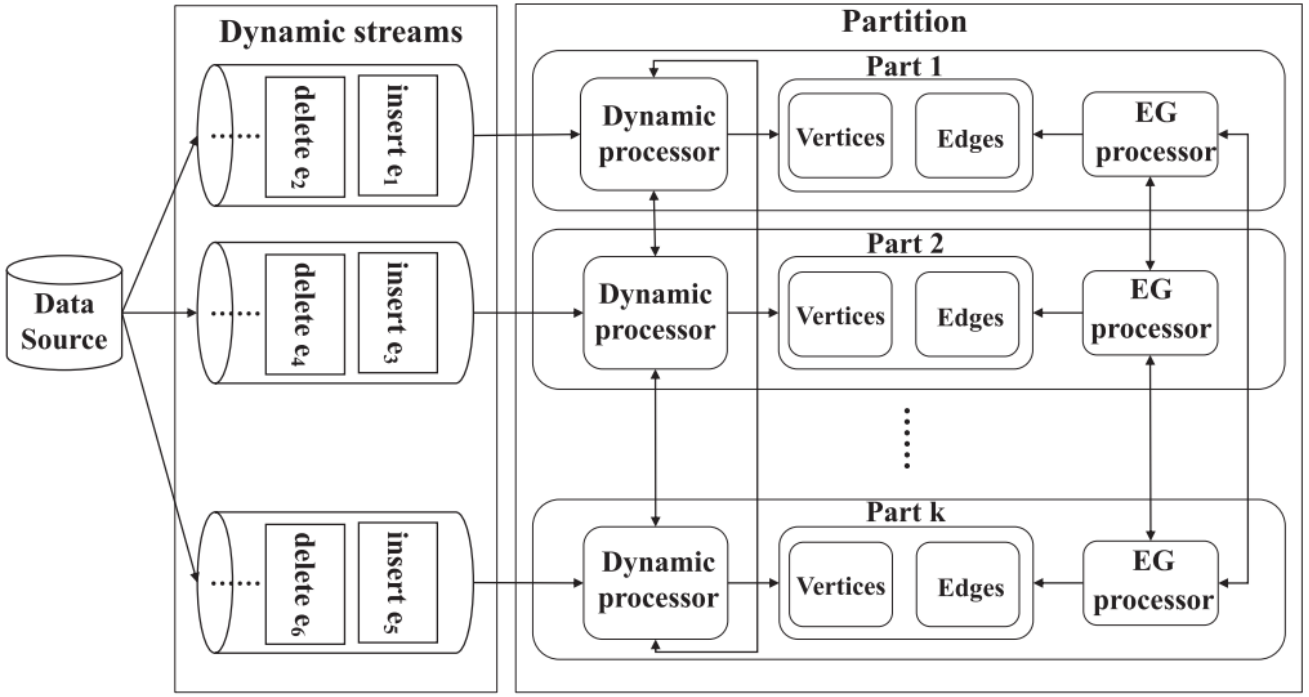
在这个例子里面EG={(a,b),(a,g),(g,b),(g,c)}
evc={a,c,b}
ivc={a,c}
迁移之后的gain就是1



GR-DEP 动态边划分算法——边组重分配

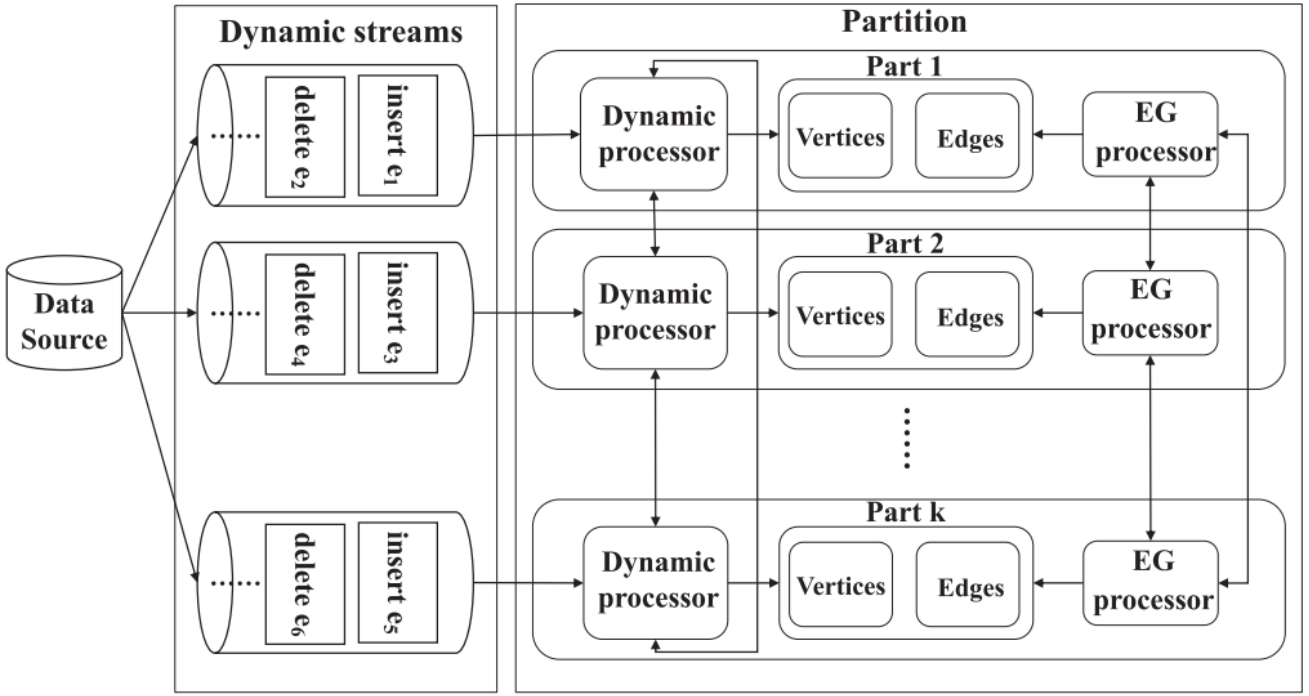
如何找到EG并且如何迁移EG是算法的核心。

- GR-DEP的结构如下图所示
- 把图的动态变化从一个分布式流里面进入划分。
- 每个子图（part）都有一个动态处理器(Dynamic processor)和一个EG processor。每个点都记录了其邻居点的列表以及它所有副本位置的集合。



GR-DEP 动态边划分算法——边组重分配

Dynamic Processor： 处理图的动态变化
对于删除一个边：首先在当前子图找到端点，如果端点不在当前子图里面，这个操作将会被发送到任意一个拥有这个端点的子图，然后这两个端点都将被在彼此的邻节点列表上互相删除。如果一个点的邻接点列表为空，则将这个点删除。
对于插入一条边：HDRF算法



插入边——HDRF算法

在插入边的时候要考虑两个因素:

- 子图划分的平衡性
- 点的副本尽可能小

HDRF算法对每一个子图（part）评分，按以下公式，选择得分最大的区域插入边。

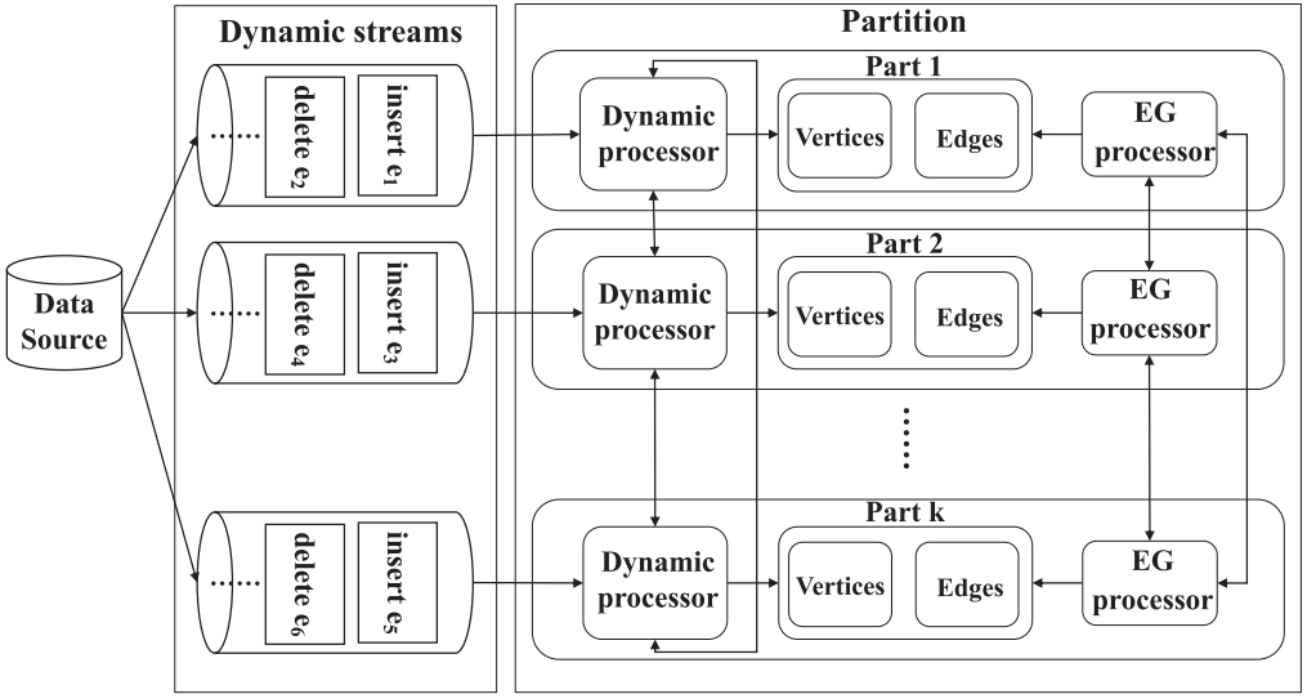
$$C(v_i, v_j, P_s) = C_{REP}(v_i, v_j, P_s) + C_{BAL}(P_s),$$

CREP表示Ps里面Vi,Vj的副本个数。
Cbal表示Ps剩下的负载空间。

GR-DEP 动态边划分算法——边组重分配

EG Processor： 找到在动态变化的局部区域的EG
因为找一次EG的开销很大，所以每次图动态变化后都寻找一次EG是不现实的。事实上，在找到一个EG，并且重新分配之后，就可以假设局部结构是最优的，即下一次不可能找到一个新的EG。

作者标记那些已经找到了EG的端点，并用一个参数 T 控制搜索每一个标记点的次数，因为只需要每T次找一下EG即可。



Algorithm 2. *EG* Processor

Input: Edge e, parameter T

1: **if** the endpoint of e appears in the dynamic change for the first time or the Tth time after reassignment **then**
2: search *EG* near e;
3: **if** an *EG* is found **then**
4: migrate *EG* to its destination part;
5: **end if**
6: **end if**

GR-DEP 动态边划分算法——边组重分配

Structure-Aware Search Method——结构感知搜索方法:

首先EG的结构是不固定的，所以一个固定的搜索方法（DFS or BFS）无法找出EG。
作者提出了一个结构感知的优先级搜索方法。这个算法在遍历点的过程中逐步找出EG。每个遍历到的点会计算一个得分score，表示这个点对于EG的贡献程度。算法描述如下：
首先低于最低负载的子图（part）是不能迁移组出去的。（因为这会导致负载均衡）
然后假设动态变化是边（Vi，Vj）插入（or删除）到Ps。
首先从端点处开始搜索，初始化一个优先队列Q，集合VG表示已经选择的点，集合EG表示已经选择的边。
在整个搜索过程中，保持记录两个数值evc和ivc。
只要队列Q非空就搜索就不停止，每次取出队头的点p（这个当前有最高的优先级），然后把p和VG中相连的边都加入EG，再把p加入集合VG。然后更新evc和ivc。
如果最大的gain是正值就停止搜索。取得最大正值的Pd，就是EG要迁移的子图。否则将p的邻居节点加入到Q继续迭代。

GR-DEP 动态边划分算法——边组重分配

Structure-Aware Search Method——结构感知搜索方法：

Algorithm 3. Structure-Aware Search Method

Input: Vertex v, part P_s, parameter M_1;
Output: EG;

if $|P_s| < (2-\varepsilon)\frac{|E|}{k}$ **then**
  return null;
end if
priority queue $Q = \{v\}$, $EG = \emptyset$, $VG = \emptyset$;
$evc\_num\_arr = \{0, \ldots\}$, $ivc\_num = 0$;
while $Q \neq \emptyset$ & $|EG| < M_1$ **do**
  $v_q \leftarrow Q.pull()$;
  push all edges between v_q and VG into EG;
  $VG \leftarrow VG \cup \{v_q\}$;
  update evc_num_arr and ivc_num;
  for all other parts $P_d \in P$, calculate $gain_{s\to d} = evc\_num\_arr[d] - ivc\_num$;
  $MaxGain \leftarrow \max_{|P_d| < \varepsilon\frac{|E|}{k}} gain_{s\to d}$;
  if $MaxGain > 0$ **then**
    return EG;
  end if
  if v_q is not a high-degree vertex **then**
    calculate priority for all neighbor vertices of v_q and push them into Q;
  end if
end while
return null;

GR-DEP 动态边划分算法——边组重分配

Structure-Aware Search Method——结构感知搜索方法:

搜索中的优先级如何计算?

这个优先级的值需要暗指出构建EG的信息。根据公式 $gain_{i\to j} = |evc| - |ivc|,$

要获得更好的效果，就要是gain值越大，即增大evc或者减少ivc。

所以可以增大evc或者减少ivc的点有更高的优先级。

所以对于一个点v有两种情况可以对计算EG有贡献

- v是一个复制点，可以增加evc的值
- v是使得在VG中的邻居节点不在是一个ivc，所以就可以减少ivc的值

得出权值计算公式

$$Priority(v) = (|R(v)| - 1) + \sum_{u \in N_s(v) \cap VG} \frac{1}{|N_s(u) \backslash V(EG)|},$$

GR-DEP 动态边划分算法——边组重分配

Egocentric Search Method:自我中心的搜索算法

结构感知的搜索方法可以根据局部结构的特点使得搜索出EG的可能性最大化，然而还是存在以下问题。

- 对于子图内部的改变，搜索成功率低
- 对于度数较高的边界点，搜索开销很大

对于以上两种情况，提出了一个中心搜索算法。它指定一个边的每个端点都只能在以自我为中心的网络中寻找EG。

自我中心搜索算法只在一个固定的小区域里面搜索EG，可能会漏掉一些，但是开销较小并且稳定。所以适合中心变化，以及度数较高的点。

GR-DEP 动态边划分算法——边组重分配

Egocentric Search Method:自我中心的搜索算法伪码

Algorithm 4. Egocentric Search Method

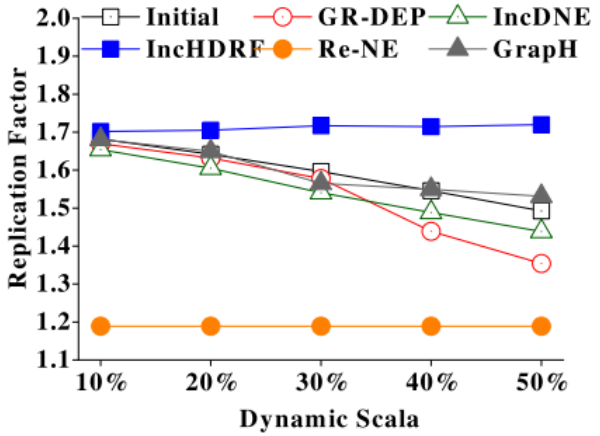
Input: Vertex v, part P_s, parameter M_2;
Output: EG;

if $|P_s| < (2-\varepsilon)\frac{|E|}{k}$ **then**
  return null;
end if
$evc\_num\_arr = \{0, \ldots\}$, $ivc\_num = 0$;
$EG = \emptyset$, $VG = \{v\}$;
if $|N_s(v)| > M_2$ **then**
  \\cr $N_s(v)$ is the set of neighbor vertices of v in P_s;
  Add the M_2 vertices with the most replicas in $N_s(v)$ into VG;
else
  $VG \leftarrow VG \cup N_s(v)$;
end if
for each v in VG, update evc_num_arr by $R(v)$;
$P_d \leftarrow \arg\min_{|P_d| < \varepsilon\frac{|E|}{k}} evc\_num\_arr[d]$;
remove the vertex v that $P_d \notin R(v)$ from VG;
check if the remaining vertices in VG are ivc, and update ivc_num;
if $evc\_num\_arr[d] - ivc\_num > 0$ **then**
  add all edges between vertices in VG into EG;
  return EG;
end if
return null;

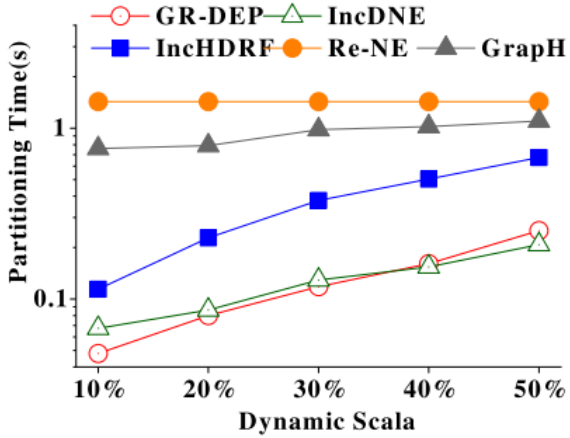
需要遍历VG3次：
第一次找出VG中点割最多的part，作为目标part。
第二次移除VG中在目标part没有副本的点
第三次计算VG中ivc的数量
如果gain值大于0，将VG中所有相连的边加入到EG

实验结果

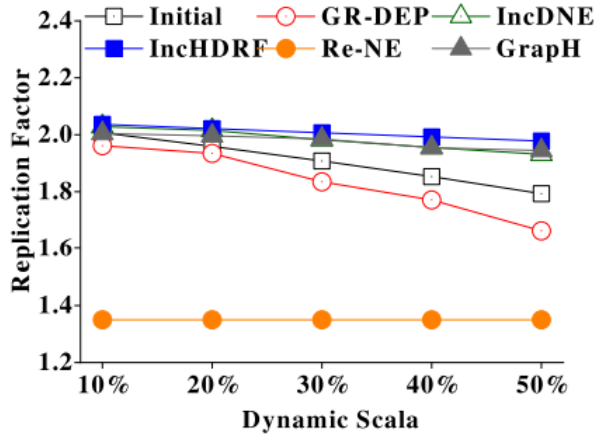
实验表明，提出的算法的划分效率和副本因子均高于最先进的算法



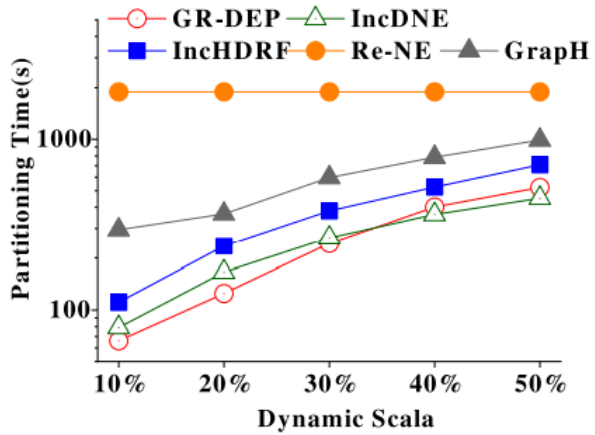
(a) Replication factor (DBLP)



(b) Partitioning time (DBLP)



(c) Replication Factor (Twitter)



(d) Partitioning time (Twitter)

Thanks

The user can demonstrate on a projector or computer or print the it into a film to be used in a wider field