摘要

随着 Web 2.0 时代图数据可用性和规模的提高，图划分成为了平衡计算工作量，效率较高的预处理技术之一。由于划分整个图价格高昂，因此最近出现一些尝试性的措施，面向流图划分技术，使之可以快速运行，容易类比，并且增量更新。遗憾的是，实验表明：鉴于在超步过程中访问模式工作量的不同，每个划分部分的运行时间仍旧难以平衡。除此之外，在对图的局部视图的算法，一次遍历流划分所得到的结果并不总是令人满意。
在这篇论文中，我们提出 LogGP，一种基于日志的图划分系统，可以进行记录、分析、重复使用历史数据信息改善划分结果。LogGP 可以被用作中间件，也可以轻易地被部署到许多最先进的图加工系统中去。LogGP 利用历史划分成果来运行超图，用一种较为新颖的超图流划分途径来运作一个更好的初始流图划分结果。在执行过程中，系统利用运行日志优化图标划分，这可以有效避免（系统）性能的劣化。更多地，LogGP 还可以适应结构变化，进行动态的再划分（调整）。在一个中等规模的计算集群上用真实的表格数据进行广泛实验证明了我们方法的优越性，堪比最先进的解决方案。

1.介绍

数据划分已被研究数十载。近来，随着互联网数据规模日趋变大，数据划分，尤其是图的划分已经吸引了越来越多的注意力。来自网页前所未有的数据增殖需要高效的处理手段应对不同的工作量。许多类似的框架工作已被推荐用来处理大规模图，例如：

Proceedings of the VLDB Endowment, Vol. 7, No. X

)

Pregel, GraphLab and PowerGraph。正如其他分配系统，图数据划分是向外延展计算能力的关键技术。
Pregel,作为其中一个有代表性的系统，由谷歌公司研发，基于BSP（bulk-synchronous parallel）模型，并采用一种“以点为中心”的理念，在一系列的超步中每个点执

行一个用户定义的功能（UDF）。通过预设，Pregel 运用哈希函数分配点。尽管哈希划分在全部已分配的计算结点生成许多均衡性良好的点，许多消息已表明：在整个结点范围内更新会导致巨额的信息通信成本。
因此，一些划分方式在类-Pregel的系统上被提出，它们大多数依托k-平衡 图划分（k-balanced graph partitioning [10]）。k-平衡图划分旨在计算结点和平衡每一划分部分之间最小化整体交流成本。Andreevet al[6]证明k-平衡划分是NP-Hard类问题。已经提出了几种近似演算法和多级探试算法。但是，随着图的尺寸增大，它们都受困于划分时间的增加。正如在[25]中所展示的那样，多级途径需要超过8.5小时用来从Twitter上划分约15亿条边的图，有时候这时间甚至会超过花在处理工作量上的时间。因此最近，一些工作专注于更简单的、划分耗时更少的流探试用于获取与多级探试可比的结果。[23,25]尽管划分结果在每个结点中平衡顶点个数、减少信息交流成本，对许多图工作负载，并不是每一超步中每个顶点都会运行UDF，所以对某些工作而言，流或多级图划分算法仍将会遇到不准确的运行时间。

- 为了解决这个问题，几种方法最近被提出：Yang et al.[27]提出一种适应工作量变化的动态复写。Shang et al.[21]则研究几种图算法，提出简单有效的法则，获取工作量的动态平衡。然而，这些方法需要重新准备图应对无论何时的新工作运行其上，这没有在划分成果上有什么改进。事实上，运行的数据和历史划分日志能够给我们提供有用的信息改进划分结果。在这篇论文中，我们研究怎样记录、分析、重复使用这些数据和日志，改善图的划分结果。我们解决了运行时间的不平衡问题并通过改善图划分质量，减少了工作的运行时间。我们开发了一中新颖的图划分管理模式-LogGP。可以重复使用之前的和正在运行的数据信息改善划分。
LogGP 有两项基于日志(或记录)的新颖技术。LogGP 首先组合图和历史划分结果生成一个超图，并基于超图，流从而获取更好的初始划分结果。当工作被执行时，LogGP 再用运行数据估计每个结点的运行时间，在下一超步重新分配以减少超步的运行时间。为估计每一超步时间，一种创新的描述工作及图轮廓的技术被提出。LogGP 可被用在中等规模并且可以轻易部署在类似 Pregel 的系统上。
- 我们在 Giraph（——Pregel 的一种开源版本）上实施 LogGP。LogGP 用在不同工作量上的数据集验证其表现。实验结果表明这种被推荐的划分方法表现优于已存在的流方法，证实它具备优越的性能。

我们在这篇论文中的贡献可以被总结如下：

1. 我们确认了一个在大规模图处理系统中比较重要的运行时间不平衡问题；
2. 我们设计并实现LogGP，重复使用之前和正在运行的数据信息。用于划分改善，推出超图再划分和超步再划分技术。
3. 我们进行广泛实验展示我们方法的优势。

论文的余下部分组织如下:第2部分，回顾图划分问题及其相关表现问题。在第3，4部分，我们呈现新颖的超图再划分（Hyper Graph Repartitioning）和超步再划分技术（超步 Repartitioning）。接下来是第5部分LogGP的体系架构。第6部分报告了大量实验研究的发现。最后，我们在论文的第7,8部分介绍相关工作和总结全文。

2.背景

在这个部分，我们首先介绍Pregel系统并将我们的系统原型建立其上。继而我们将介绍图划分和流图划分问题。最后，分析在Pregel下分析图划分问题。

Pregel是一个分布式图处理系统，由谷歌提出。基于BSP模型[26].在BSP模型中，图处理工作通过许多由同步障碍所划分的超步 被计算。在每一超步中，每个工作者，或称结点，执行用户定义的功能，相对在其上的 顶点子集以异步方式计算。这些点被称作**活跃点**。顶点的其余部分，那些不被应用于超步中的点，被称为**非活跃点**。除此之外，结点为下一超步给其临近点发送必要信息。一旦交流和计算完成，就会有一个全局同步障碍保证所有结点准备好下一超步或分配任务完成。

和其他分布式系统一样，为获取最佳表现，工作量应该平等分配每一结点，以最小化交互成本。因此，图划分是一项获取更好量测性必不可少的技术。

图划分：我们现在正式描述一般的图划分问题。我们用G=(V,E)表示将被划分的图。V是点集，E是边集。图可能是有向的，也可能是无向的。令 $P_k = \{V_1,...,V_k\}$ 为V子集k的一个集合。P_k 作为G的一个划分。如果：$V_i \neq \emptyset, V_i \cap V_j = \emptyset$, 并且 $\cup V_i = V\ (i, j = 1, \ldots \ldots, k, i \neq j)$. 我们称P_k的元素V_i为划分的部分。数字k被称作划分的基数。在这篇论文里，我们假定每一个V_i被分配一个计算的结点，用V_i来指代那个结点中的点集。

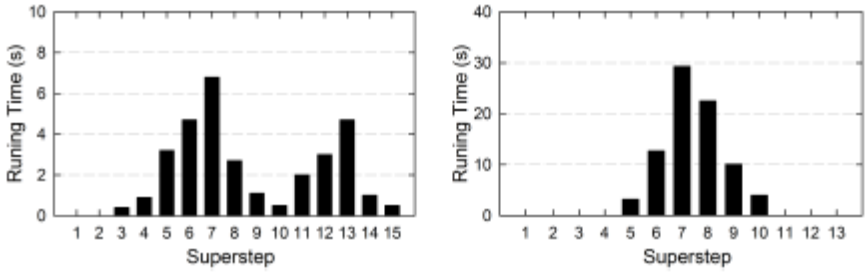
图划分问题在于基于一个客观的函数找到一个可选的划分P_k 。这是一个组合选择问题。定义如下：

定义一.图划分被定义三重序列（S,p,f）,S是G所有划分中的一个离散集合。p基于S生成S子集，被称作可取的方案集：S_p ，所有在S_p中的划分可从p中获取。f是目标函数。图划分问题旨在找到一个划分$\overline{P}$， $\overline{P} \in S_p$ ，最小化f(p)：

$$f(\bar{P}) = min_{P \in S_p} f(P) \qquad (1)$$

一个简单的策略是使用哈希函数划分数据，这被作为预设策略应用于Pregel[17]中。但是，这种方式造成很高的通信成本。因此，使其表现逊色不少。一些工作用近似值或多级方式划分图。但是，随着图越来越大，划分费用依旧难以承受[25]。因此，最近的一些工作[23,25]使用流划分探索法划分图。

尤其是如果图的点和临近点集以某种顺序排列，我们将基于点流划分图。这叫做**流图划分算法**。流图划分算法决定哪一部分安排给哪一个进来的点。一旦该顶点被布置好，它就不会被清除。



a. SSSP b. BFS

Figure 1: Running Time of Supersteps on a Node

这些k-平衡图划分方式顶点计算工作，而每一部分的通信时间不被考虑。因此，每个结点的运行时间（既包含计算时间，也包含通信时间）可能被歪曲。一个不平衡的理由在于Traversal-Style 工作负载[21],例如：SSSP和BFS。这些工作在不同的超步内每一结点上浏览不同的活跃点，这在初始划分时很难预测。因此，每个结点在每个具体超步上的计算时间和通信时间是不平衡的。数据1展示了以工作量SSSP和BFS用多级图划分算法进行初始划分的运行时间。我们可以看到，在每个超步中都会有一些有意义的运行时间上的不平衡。对于总是活跃的工作量[21],也会有运行时间的不平衡，这是因为k-平衡图划分仅仅平衡顶点的计算工作，而对于通信时间则不予考量。

对类-Pregel系统，结点运行时间上的不平衡影响整个图处理工作的运行时间，限制了整个系统的测量。因此，使用静态初始划分结果，生成结果没有分析工作的行为，不能够平衡每个结点的运行时间。

为缓解上边提到的问题，我们提出一种图划分的课题 LogGP，它可以开发历史图信息。LogGP运用两种新颖的技术——超图再划分技术和超步再划分技术，在下两部分将被详细讨论。

3.超图再划分

下面我们将首先介绍怎样借助历史日志生成更好的初始划分结果。正如部分1中提到的，多级划分算法对大图（划分）速度太慢。因此，我们使用流方式生成初始划分结果。

令$P^t = V_1^t, \ldots, V_k^t$为时刻t下的一种划分结果，$V_i^t$为划分i在时刻t下的点集。A流图划分是连续呈现点v和它的临近点N(v),并且使用流探试来分配v给划分i，只利用了包含在当前的划分P^t内的信息。尽管一次(one-pass)流划分算法时间更短，结果却并不如多级解决方案好。因为它只有图中已分配的点的信息。在这篇论文里，我们用历史划分日志在运行流图划分和提炼初始划分结果时增强可见性。

我们使用两种由历史划分结果提供的信息。

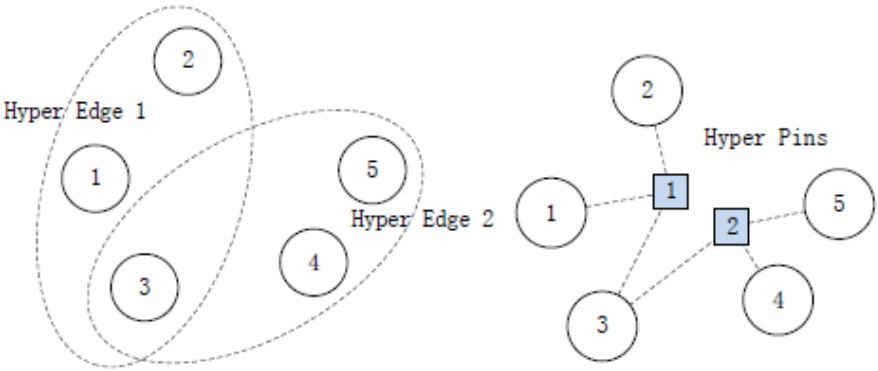
第一种是相同图或相同部分的最后一次划分结果，这结果已经基于流划分算

法将图划分为几部分。尽管没有被优化，相对随机图来说，它仍给我们提供了一个更好的输入结果。我们尝试合理使用这个最终划分结果，以之为流图划分提供比包括在现有划分中点更多的信息。

另一个信息是在执行先前工作已评估的活跃点集，被称为**记录活跃点集**(LAVS),一个LAVS是一个活跃点集，为在连续超步中的结点。我们发现在连续超步中的活跃点彼此之间有内部的联系。尤其是当工作中已被评估的点有自然联系。例如：Semi-clustering[17] Maximal IndependentSets [16] and N-hop Friends List [7].这些工作量通过内部联系将点自然聚集在一起。以Semi-clustering为例，一个半群(semi-cluster)在一个共有的图中是一群彼此交易的人，和其他人联系相对较弱。当在图上运行Semi-clustering，这个工作负载的LAVS连通顶点(代表人),彼此之间有很强联系。因此，如果使用LAVS作为图划分附加信息的运行时间，我们一开始就知道这个图的联系可以帮助我们的流算法决定哪些点需要被放在一起。为控制在不同工作环境和图形下，LAVS中点的数量，我们使用一个参数k决定多少连续的超步LAVS将会相关。具体怎样记录和计算LAVS的细节将在第5部分被讨论。

3.1 超图

为组合这两种用 原图 得到的历史信息，我们使用超图 代表有用的历史信息和原图结构。超图H=(V_h, E_h)是泛化的图，它的边可以连接超过两个顶点，称为超边(也叫网)。超图含有点集 V_h和超边集E_h。每条超边都是V的子集。例如，在表2(a)中展示的。有两条超边呈现为虚线圆圈。第一个(超边1)包含点V_1, V_2和V_3。另一个(超边2)包含点 V_3, V_4和V_5。另一个呈现超边的方式是转化超边为同等的**超边标签**和**超针** ，如表2(b)所示。超针和超边标签可以被看做是简单图的虚拟顶点和边。超边e_h的参数|e_h|是它包含的顶点个数。如果每个超边都有参数k的话，那么超图H是k-regular 。事实上，简单图是2-regular超图。超图自然地表示我们信息的三个布局：原图，最后划分结果和LAVS。我们接下来着手接收怎样形成它。



a. Hyper Edge b. Hyper Pin

Figure 2: Two Ways to Represent Hyper Graph

怎样形成超图：假定原图为G=(V_0, E_0),我们使用同样的点集V_0形成超图点 $V_h = V_0$.

超图E_h的边集由3部分组成：

1. 我们使用v_s和v_e指代边集E_0中的原图边的两顶点。然后我们向E_h添加一条超边$h_e = \{v_s, v_e\}$。

2. 我们用$P_{last} = (P^1_{last}, \dots, P^n_{last})$指代图的最后划分结果。$P^i_{last}$指代第i部分的划分结果。我们向$E_h$添加超边$h_e = P^n_{last}$。

3. 我们用$LAVS_i = LAVS^1_i, \dots, LAVS^i_m$指代LAVS结点集，带有全部的m数，并向$E_h$添加每一条超边$h_e = LAVS^i_m$ 。

数据3展示一个例子，关于怎样生成一个超图。我们首先改造原图(数据 3a)成超图边(数据3b)。这些超边$h_e = \{v_s, v_e\}$被标记为超针“1”。然后我们包括了图最近的划分结果(数据3c)。这些超边$P_{last} = (P^1_{last}, \dots, P^n_{last})$被标记为

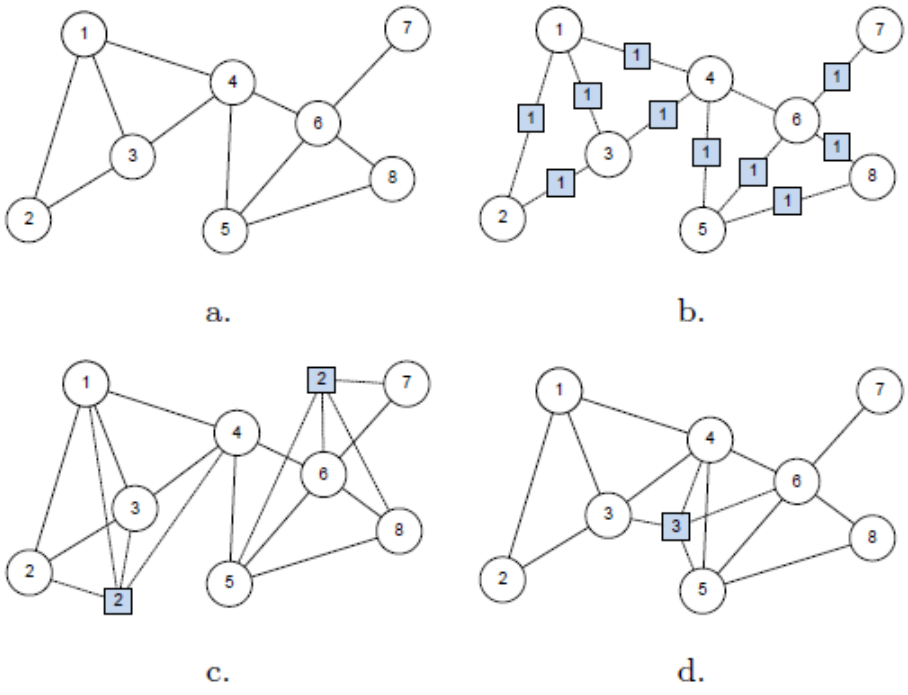


Figure 3: Example of Hyper Graph Generation

超针”2”。最后，我们添加LAVS得到最终超图，标记为超针“3”。

生成超图之后，我们设计一种新颖的超图流划分技术(HSP)。我们首先转化超边为同等的**超边标签**和**超针**。超图$H = (V, E_h)$）也可被定义为三重序列$H = (V, \overline{H}, \overline{P})$,V 是原始点集，$\overline{H}$是超边标签集，$\overline{P}$代表连接V的元素和$\overline{H}$元素的边集。

正如数据2所示，我们可以使用超边标签和超针(数据2（b）)表示超边(数据2(a))。在这些转化后，现在我们将把超边作为顶点(超针)，并使用它们进行一次(one-pass)流图划分。我们仅仅生成了参数不超过2的超边生成了超针。因此，原始边不被表示成超针。以这种方式，超针和超边标签提供了包括之前划分结果和活跃点日志在内的两方面顶点连接信息。

3.2 超图再划分

我们的超图再划分方法由两部分组成。首先，我们将超针分配到不同的划分区域，并将原始顶点也分配于此。超边标签被看做是一条边，被用于流划分算法，作为附加边。当超针被分配到一片区域，在相同超边区域的点更有可能被分配到此分区。超针实际上提供了给流算法几种全局的视角。超边可以呈现几种点的内部联系。

流图划分算法基于不求完整的信息作出决定，数据的顺序将有效硬性表现[23]。点顺序以几种方式存在不同。如，随意排放，BFS/DFS从起始点或对立顺序。Isabelle[22]证明BFS和DFS作为一种随机方式生成几近相同的结果。在这些顺序中，随机顺序是最容易保证大规模数据脚本。并且为了简易性，我们可以限制我们的超图再划分，只考虑随机顺序的结点。因此，我们随机安排超针，紧随其后是原始顶点分配。

我们使用一个流加载器从超图读点。然后加载器会向正如算法1所展示执行流图划分的程序发送顶点及它们的临近点。探试函数基于当前划分状态分配进来的点并输入图。

Algorithm 1 Hyper Graph Repartitioning

```
Input: Hyper Graph H
Let P_k = V_1, V_2, ..., V_k be the current partitioning result sets.
S ← streamingloader(H)
generate random order to S
for each hyper graph pins p̄ in S do
    index ← HeuristicFunc(P_k, p̄);
    Insert hyper graph pins p̄ into V_index;
end for
for each vertex v in S do
    index ← HeuristicFunc(P_k, v);
    Insert vertex v into V_index;
end for
```

在[23,25]中，一些在广泛范围里探试流结点分配表现的工作被提出。在这篇论文里，我们不关注推出新的流探试法。事实上，我们的方法可以适应这些探试法。由于LDG在这些算法中有了最佳表现，这里我们以Linear Deterministic Greedy(LDG)作为代表。在LDG中，每个点v被分配到划分区域基于：

$$index = argmax|V_i \cap N(v)| \left(1 - \frac{|V_i|}{C_i}\right) \qquad (2)$$

C_i是分区i的最大容量，N(v)是v的临近点集。

正如我们上边提到的，每次使用图都应该轻微改变。事实上，图的数据集是典型的动态的。也就是说，图的结构随时间改变。例如，Twitter图可能有数十亿个点和边每秒没频率都在改变。因此考虑我们系统容纳增加图的能力尤为重要。超图再划分的一个优势是它不必由太多修订：我们的方法可以用历史信息重划分新图，也可以通过超图再划分将新点作为原始点分配。

4.超步再划分

在这一部分，我们分析在超步中每一结点在运行时间上的不平衡，并基于执行时收集到的数据提出一种全新的再划分策略。

4.1 工作运行时间

为改进图加工工作的表现，我们首先尝试在Pregel系统观察模拟工作运行时间。我们将分析算法并模拟出每一超步产生的影响。除了将算法分成不同种类[21],我们想找到一种一般的方式在图上模拟这些算法。

在BSP模型中，一项工作被分成连续的超步，正如数据4 中所展示。在每一超步内，结点彼此交互。也就是说，发送/接收信息到/从它的临近点。

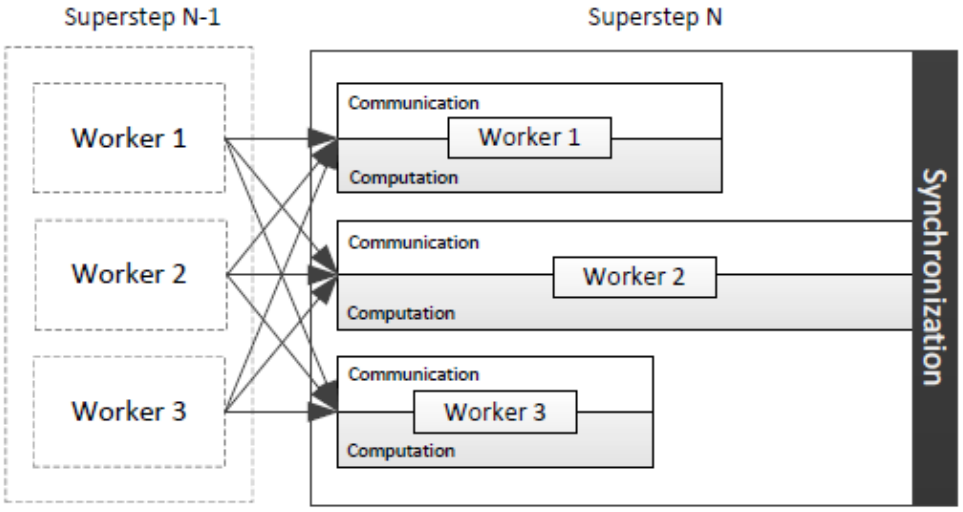


Figure 4: Abstraction of the BSP Model

在所有点完成计算和交互工作后，将会有一个障碍(barrier)保证结点为下一超步准备好。当所有的活跃点是空的，或最大数量的超步抵达，图算法执行会被终止。

每一超步的运行时间由最慢的结点决定。令ST_i^n为结点$Node_i$花在第n超步上的时间。同步障碍在每一超步上的时间花费大抵相同，对比计算和通信时间微不足道，所以我们在这篇论文里忽略它。然后，图算法的全部运行时间，JobTime可以通过下式计算：

$$JobTime = \sum(max(ST_i^n)) \tag{3}$$

图划分技术帮助图处理系统最小化JobTime。如在第2部分提到的，时间花销取决于通信时间和计算时间两项：

$$ST_i^n = f(Tcomp_i^n, Tcomm_i^n) \tag{4}$$

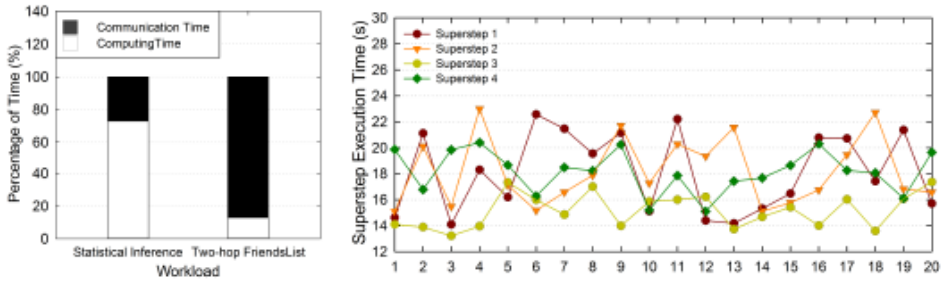
在这里T $comp_i^n$和T $comm_i^n$分别指代$Node_i$在第n超步中的计算时间和通信时间。函数f(x,y)取决于系统的实现。如果系统采用一个I/O闭塞模型，在其内，CPU和I/O连续执行。我们可以将时间加起来，这意味着f(x,y)=x+y.如果系统类似处理I/O操作，超步的运行时间ST_i^n取决于较慢的那个。因此，f(x,y)=max(x,y)。我们的系

统基于I/O闭塞模型，然而当用一个I/O类似的系统时同样的结果可被获得。在我们的系统内，第n超步的运行时间和JobTime可以被表示为：

$$ST^n = Max(Tcomp_i^n + Tcomm_i^n) \quad (5)$$

$$JobTime = \sum Max(Tcomp_i^n + Tcomm_i^n) \quad (6)$$

由于$T\ comp_i^n$和$T\ comm_i^n$依赖图的算法和硬件，在图的初始划分语句中获取这些信息非常困难。数据5(a)演示了在两种工作负载上的一次实验 Statistical Inference 和 Twohop Friend List.我们记录平均时间比，以此在Giraph执行它们时计算UDF函数和发送/接收数据。正如我们所看到的，Statistical Inference的运行时间由计算工作时间决定，而Twohop Friend List的则由I/O数据传送决定。传统的图划分算法旨在最小化边的切割而不是运行时间上，因而会忽略这些因素。结果是，对于每一超步，运行时间都有可能会被曲解。正如数据5(b)所展示的那样，我们记录了Pagerank工作负载在20个结点簇上的前4个迭代器。运行时间也会有显著的不同。



a. Workloads b. Nodes

Figure 5: Running Time of Superstep on Different Nodes and Workloads

4.2 再划分的探试

传统数据库系统用收集到的历史数据估计运行时间，用于问题优化。对类似的图处理系统，为解决不平衡问题，在算法执行期间，我们收集有用的信息在估测不久之后的时间和动态重分配顶点之间优化工作负载，实现平衡。我们将首先讨论怎样估计运行时间。

我们收集数据信息估测每一分区的运行时间。正如4.1处所提，超步的运行时间等于计算部分和通信部分用时之和。

$$ST_i^n = Tcomp_i^n + Tcomm_i^n \quad (7)$$

计算时间T $comp_i^n$由整个活跃点集决定，设为A_i^n,并且用户定义函数。这是因为活跃点集将执行UDF。而通信时间，由活跃点集的部分引发。如在数据6所示，我们将分区内的点分成3种形式。虚线指代在不同分区两顶点的联系，实线表示点在相同的分区。

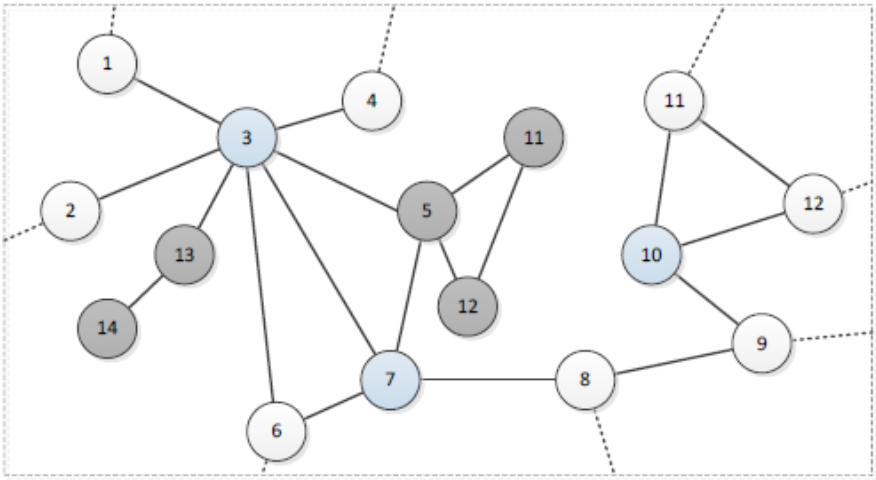


Figure 6: Three Types of Vertices in a Partition

类型1：对相同分区内的点来说，点和它们的临近点是毗邻的。例如：点5，11,12,13，14都是这种类型。我们用 α_i^n 指代在第n超步中结点i上的这种类型的点。

类型2：点与不同分区的点毗邻。例如：点1,2和4都是这种类型。我们用β_i^n指代在第n超步中结点i上的这种类型的点。

类型3：点只和相同分区点毗邻，它们的临近点毗邻其他分区的点。我们用γ_i^n指代在第n超步中结点i上的这种类型的点。点3,7,和10都是属于这种类型。

第一种类型的点只生成本地消息，在相似的图处理系统内，提供、处理本地消息要远快于没有本地消息。因此，类型1的点不生成通信时间。

第二种类型的点从最近的分区收发信息。全部的通信时间都是由活跃点子集造成。我们现在可以表示超步运行时间为：

$$ST_i^n = tComp * |A_i^n| + tComm * |\beta_i^n| \qquad (8)$$

tComp和tComm都是每个顶点耗费的平均计算时间和通信时间。LogGP在工作执行期间使用日志获取这两项参数。详细信息将在第5部分讨论。

同类型1，类型3的点不生成通信时间，因为没有 无本地 的点临近它。但是，这些点可以在下一超步中激活类型2的点生成通信时间。因此我们可以用类型3的点估计下一超步中的通信时间。

另一个问题是我们怎样用$\boldsymbol{\gamma_i^n}$的信息得到$\boldsymbol{\beta_i^{n+1}}$的点集数。这里我们介绍一种新的度量方式，活跃比，用 λ 表示。活跃比表示活跃点 v 在超步 N 中致使该点临近点在 N+1 超步中变活跃的可能性。例如：算法 Pagerank 的活跃比总为 100%，因为活跃点总能在下一超步中激活临近点。这些参数也可以被本地日志获取。

然后，我们可以用 λ 估算$|\boldsymbol{\beta_i^{n+1}}|$和$|\boldsymbol{A_i^{n+1}}|$：

$$|\beta_i^{n+1}| = |\gamma_i^n| * \lambda \tag{9}$$

$$|A_i^{n+1}| = |A_i^n| * \lambda \tag{10}$$

第(n+1)超步运行时间可被估算：

$$ST_i^{n+1} = (tComp * |A_i^n| + tComm * |\gamma_i^n|) * \lambda \tag{11}$$

在估计下一超步的运行时间后，我们收集每个结点的预估运行时间，并且重新分配点以此重新平衡下一超步的运行时间。

然而，点的重新分配(重新划分)仍旧很困难，因为：

1) 找最优重分配是一个NP(Non-deterministic Polynomical，非确定多项式)问题，正如size-balanced图划分问题是一个NP完全问题[6]，这是我们问题的静态设定。

2) 重划分的计算性的开销一定很低。因为目标是提高应用表现，可选的技术必须是轻量级的，并且工作在大图上时可被测量。

3) 不同分区动态地同步分布式状态不可能实现。通过网络增殖全球信息招致显著的开销，这在我们的重划分技术中必须得到考量。因此，重划分只能使用图的本地视角。

基本的想法是在下一超步之前从预估划分需时较长的点到用时短的点进行重新安排。我们用临界值θ_{n+1}来决定在整个的运行时间中预估时间是否长。

$$\theta_{n+1} = \vartheta * \frac{\sum ST_i^{n+1}}{i} \qquad (12)$$

这里ϑ是我们下面将要讨论到的百分比。有些分区相较其他分区ST_i^{n+1}比θ_{n+1}大，从这些分区，我们移动点，并尝试最小化最长运行时间。
在这篇论文里，我们提出一种新颖的探试方法，在下一超步向合适分区移动那些将会生成通信成本的点。移动将来会减少整体的通信花费，并在下一超步内平衡每个结点的运行时间。对在结点$node_i$中的点v_i,LogGP 记录每个点和其他分区通信的时间，表示为$C(v_i)$。如果在结点$node_i$上需要对点重划分，我们首先会选择那些在下n+1超步可以生成远距离通信的点，记为Γ^i_{n+1}。令 N(v)为点v的临近点集。！！

$$\Gamma^i_{n+1} = \bigcap N(v), v \in \gamma^n_i \qquad (13)$$

然后我们选择远程通信次数，$C(v_i)$，对每个在Γ^i_{n+1}中非递减顺序的点，贪心用最大$C(v_t)$向有最多临近点的点分区重分配点。当一个点被清除，预估运行时间将会减少tComp+tComm。当Γ^i_{n+1}为空或下一超步的运行时间等于平均值时重分配停止。整个处理过程在算法2中被解释：

Algorithm 2 Superstep Repartitioning

1: $ST_i^{n+1} \leftarrow$ estimated running time of $N+1$ superstep
2: **if** $ST_i^{n+1} > \vartheta * \frac{\sum ST_i^{n+1}}{i}$ **then**
3: $\Gamma_{n+1}^i \leftarrow \bigcap N(v), v \in \gamma_i^n$ //$N(v)$ is the neighbor vertex set of v
4: $List \leftarrow C(v)$, v $\in \Gamma_{n+1}^i$ //$List$ records the $C(v)$ of vertex in Γ_{n+1}^i
5: $List \leftarrow$ Sort $List$ in non-descending order
6: **while** $\Gamma_{n+1}^i \neq \varnothing$ and $ST_i^{n+1} < \frac{\sum ST_i^{n+1}}{m}$ **do**
7: // m is the number of partitions
8: $v_t \leftarrow \text{pop}(List)$
9: Reassign v_t to the partition that the v_t has the most neighbors
10: $\Gamma_{n+1}^i \leftarrow \Gamma_{n+1}^i - (tComp + tComm)$
11: **end while**
12: **end if**

5. LogGP结构

在这一部分，我们概述LogGP，展示它是怎样无缝融合进系统改善表现的。我们首先提供系统结构，然后呈现系统的关键组成。最后，我们讨论怎样完成点的迁移。

系统结构：我们首先描述基本架构和LogGP的运作。数据7显示LogGP怎样与Giraph(Pregel的开源版本)融为一体，作为一个中间件。我们的系统也很容易被转移到任何基于BSP的处理系统。在Giraph系统内，有两种类型的处理结点：主人结点和工人结点。只有一个主人结点管理分配和协调工作，有多个工人结点，执行用户定义的函数以应对每个被分配给它们的结点。LogGP的组成部分运行在Giraph结点中相一致的结点上。有一个LogGP的划分管理者(PM)运行在主人结点上，为主人结点提供划分元数据。LogGP划分中介(PA)运行在每个工人结点上向PM记录、收集和报道每个工人结点的日志信息。初始图数据在PM上被划分，并被安排向每个工人结点。

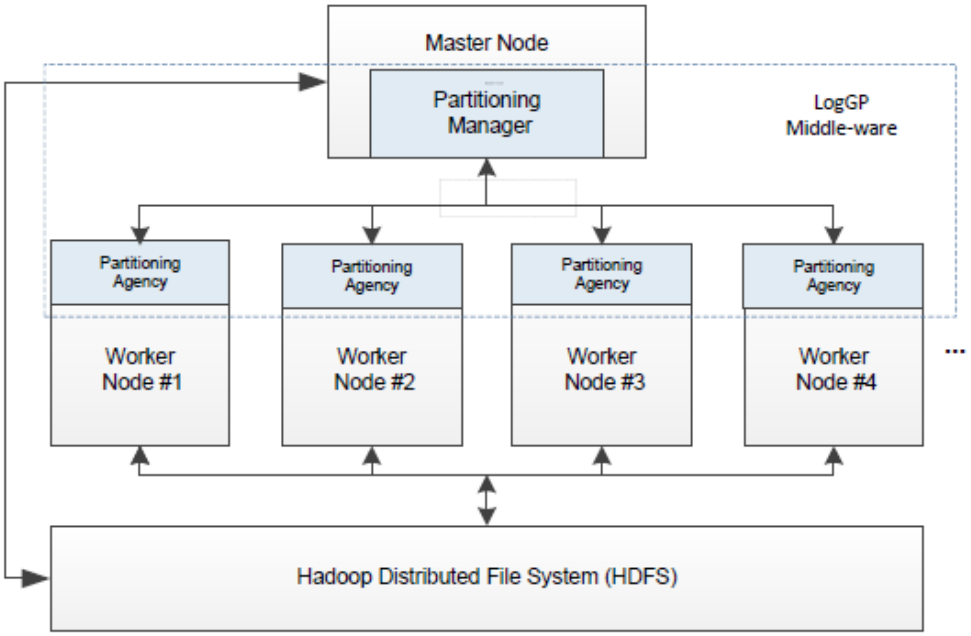


Figure 7: Architecture of LogGP Integrated on Giraph

PM和PA共享Giraph的通信框架，彼此通信。持续的数据，如历史划分结果，被存储在HDFS(Hadoop Distributed File System)[4]也共享Giraph。关于关键组成部分的详细讨论如下：

LogGP划分管理者：划分管理者(PM)是一个进程运行在每一个主人结点上。它存储图的元数据和历史划分结果。在LogGP内每个图都有一个独一无二的ID，代表一个特定的图。元数据储存图的历史信息链接，如：图的最新划分结果或LAVS。真实数据存储在HDFS，并且PM可以通过元数据连通文件。当一项工作运行在一幅图上，主人结点将会要求PM用原图生成超图，如果存在的话，会是最近的图划分结果和LAVS。这之后，PM划分图生成初始划分区域。然后PM检查元数据取得历史信息为图生成初始划分结果。然后主人结点用划分结果从HDFS向工人结点分配点。

在工作执行期间，当每个工人完成一次超步，它向PM发出划分的预估运行时间。PM从工人结点聚集、分析信息，并应用重划分策略优化下一超步的运行时间。如果在一个结点上需要一次重划分，PM将会发送一个重划分的触发，与必要的信息一起送至那个结点的PA。重划分程序将会在那个结点上开始。

LogGP划分中介：划分中介(PA)在Giraph系统中运行在每个工人结点上，收集运行信息，当在下一超步有运行时间被曲解，完成重划分。在Giraph中，PA使用相同的通信组件与PM或其他PA通信。当超步运行，PA收集信息，如tComp,tComm和C(V)，然后用这些信息计算下一超步的预估时间。事实上，当一个点正执行一个用户定义的函数，如Pagerank，PA会记录这些点的类型及相应用于计算的时间花销。当数据被发送往其他分区，PA记录点集信息，并记录用在

通信上的时间花销。在每个超步结束后，运行日志都将会被分析。并为超步生成数据日志。TComp,tComm,C(v)和其他用于超步重划分的参数都可以从这些日志中获取。当一次超步完成，PA将会向PM同步发送下一超步的预估运行时间，并等待PM的反应。如果重划分必要，为平衡运行时间，PM将会运行重划分探试。PA也会写入每一超步的活跃集的运行日志。当工作加载完成，LAVS由运行日志生成。当连续的k超步所有的LAVS被生成，结果将会被上传至HDFS，以便于PM下次工作时生成超图。这个过程很快，且不需要占用结点太多的资源。

点的迁移：对超步重划分，当动态划分课题决定从一个分区向另一分区重分配点时，三种类型的数据需要被发送：(1)最新的点值；(2)中介点列表;(3)下一超步信息。一个很合理的解决方案是，在第n超步结束和第n+1超步开始之间为点的重划分添加新的状态。LogGP使用另一种选项，在同步状态间组合点的移动。组合从一个到另一个分区发送的消息减少通信的时间花销。除此之外，需要被迁移的点数其实很少。因此，同其他操作相比，重分配开销很小。我们将进一步讨论实验研究之中点的重划分时的时间使用。

当一个点被再分配给一个新的工人结点，群中的每一个工人必须获取、存储这些信息，以便进一步向点传递信息。一个明显的选项是储存在一个由$<Vertex_{id}, Worker_{id}>$序对组成的Map(键值对)里。但是，使用这种解决方案的话，u点必须在节点间移动时向它的所有临近点广播。为减少花销，我们增加一个高效率的查表服务(将会在[24]中提及)以减少在重分配点时减少广播上的花费。

6. 评估

在这一部分，我们评估我们我们提出的划分改善方法的表现。我们在Giraph上完善LogGP[1]，和为点迁移解决方案服务的查表服务[24]。

6.1 实验设置

我们首先介绍评估的实验配置，包括数据集，评估度量和相对的方法。所有的实验都被实施在AMD Opteron 4180 2.6GhzCPU，48GB内存和10TB RAID盘的28个结点上。所有结点都被连接至1Gbt带宽的路由器上。还有一个合成图由下面的Erd•os-R_enyi随机图模型生成。这些真实世界的数据集洞可以从网页获得[5]，数据集的数据在表1 被展示。我们将它们改造成无向图，添加互补边，从原始发布版本中清除圆圈。

6.1.1 数据集

我们用5个真实的数据集：Live-Journal，Wiki-Pedia，Wiki-Talk，Twitter和Web-Google

Dataset	Nodes	Edges	Type	Size
Wiki-pedia	2,935,762	35,046,792	Web	401MB
Wiki-Talk	2,388,953	4,656,682	Web	64MB
Web-Google	875,713	8,644,106	Web	72MB
Live-Journal	4,843,953	42,845,684	Socia	479MB
Twitter	41,652,230	1,468,365,182	Social	20GB
Synthetic	5,000,000	100,000,000	Synthetic	930MB

Table 1: Graph Dataset Statistics

6.1.2 评估度量

我们使用两个度量系统评估我们实验的结果。对于这两个度量，较低的值代表了较好的表现。

边切比(ECP):它预示了图中分区之间切割边的百分比，定义为ECP=ec/|E|，ec指代在分区间切割的边数，|E|指代图中边的总数。这是一个基本度量，评估切割结果的质量。

执行时间： 我们利用两次时间花费评估系统表现。第一个是工作执行时间(JET),表示从提交工作量到完成流失的时间。基于此，我们进行评估率真实的划分有效性。划分图之后，我们在划分的图上运行工作量，并记录下时间。第二个是，总运行时间,包括工作量的执行时间，也包含图标的加载时间和划分时间。

6.1.3 对比方法

在这份实验研究之中，我们选择几个使用最新划分技术的方式，用于比较来论证我们提出方法的优势。

• 为减少图系统工作执行的整体时间，提出的LogGP方法使用两项技术，超图划分(HGR)和超步划分(SR),。为更好的检验我们方法有效性，我们也调研了HGR和SR的单个表现。

• 线性决定贪心(LDG)方法[23]被认为是最佳静态流方式之一。重流 LDG（reLDG）方法[19]被扩展用最终流划分结果生成初始图划分。

• CatchW[21]是一个动态图标工作负载，平衡随机初始划分的方法，是SR的相对方法。这两者都尝试在超步中调整划分区域。

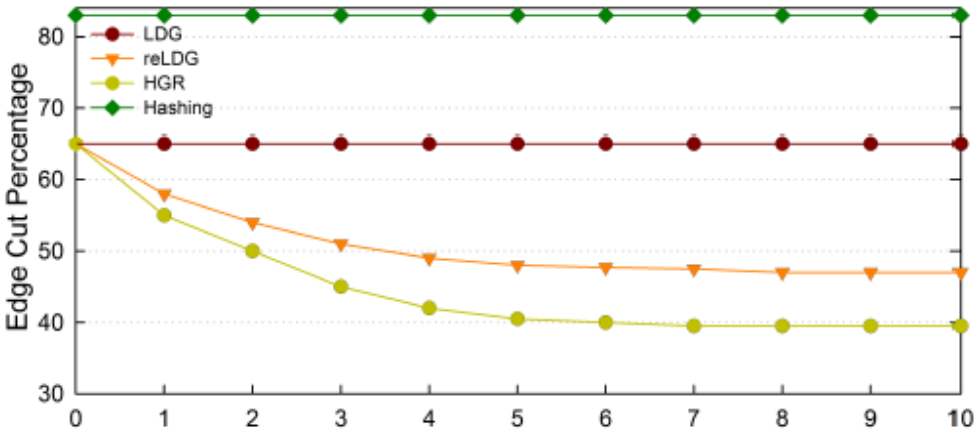
• 由于Hashing的极简性和受欢迎程度，我们也用它作为一个竞争者。例如：Pregel用默认哈希函数划分点。

6.2 超图再划分的影响

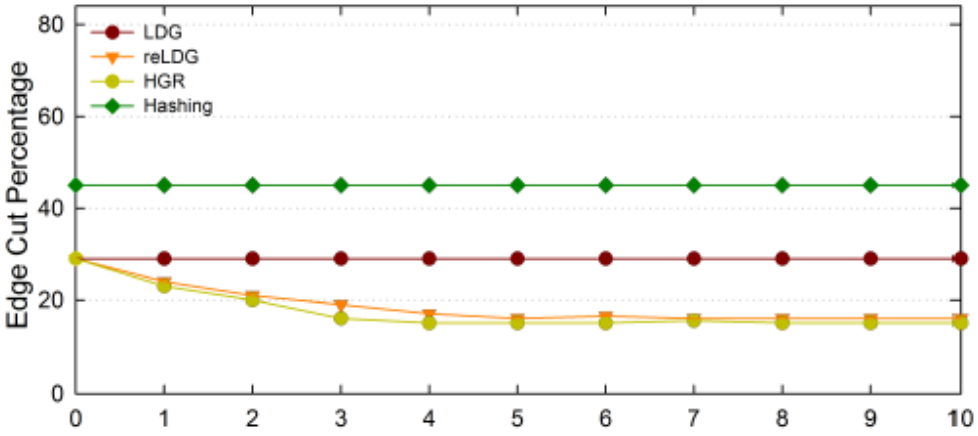
我们首先评估超图再划分(HGR)的表现。最顶尖的流算法LDG[23]，reLDG[19]和哈希法[Hashing]都被用作基本线。我们对表1展示的图的数据集。这里我们主要呈现在Live-Journal，Web-Google和Synthetic datasets(合成数据集）上的结果为Social Graph，Web Graph和Random Graph的代表。

数据8显示在这3类数据集上划分结果的ECP。每个划分算法执行10次，HGR和reHGR用最终再划分的划分结果。为了为超图生成LAVS，我们在划分之后执行了一个半群工作负载。正如所见，HGR在所有图上表现优异，尤其是在Social Graph上。这主要因为这些社交图代表真实社交网关系，具有相对较高的本地群系数。在运行完半群工作负载之后，LogGP记录这些点内部关系的半群的LAVS。然后HGR用这些再划分图的附加信息生成超图。在超图的帮助之下，流算法HGR可以使用更多有用相关点以获得更好的划分结果。因此，我们的方法优于reLDG，它仅仅使用划分结果，而我们也是用附加的LAVS信息进行初始划分。对于Web Graph ，广为所知按幂函数定律曲解，我们的方法仍旧表现最佳。尽管在Synthetic Random Graph(随机合成图)之中，我们的方法取得了最小的改善，对比LDG，ECP仍旧减少约15%。对Random Graph(随即图)，点与点之间没有内部联系，因此，LAVS不能取到关系，导致进步较小。但是，对这些随机的图(注:有的译为随机曲线)显示之前的划分结果很有帮助。

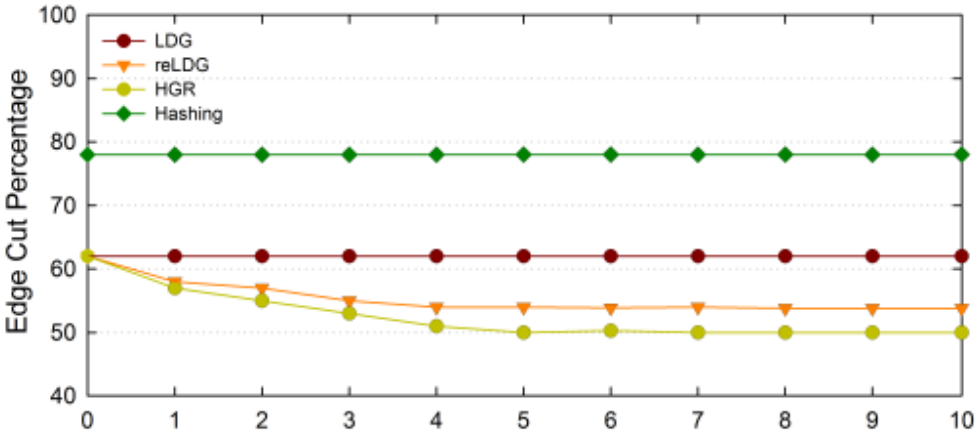
除此之外，在10次再划分后我们评估了这些划分结果的运行时间。我们在Live-Journal数据集上运行了Pagerank的10-iteration 和Semi-Clustering的工作负载。比较了用Hashing，reLDG，LDG和用 HGR的运行时间。如数据9(a)所示，在这两项工作量上，HGR显著减少了运行时间，这确认了我们的方法可以在用超图时改善流划分结果。



a. ECP Results of 10 Iterations on Live-Journal



b. ECP Results of 10 Iterations on Web-Google

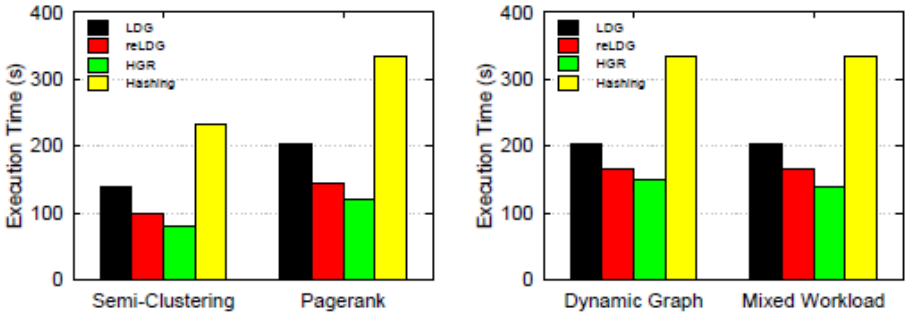


c. ECP Results of 10 Iterations on Random Graph

Figure 8: ECP of 10 Iterations on 3 Types of Datasets

在真实世界，工作量和图随时间而变。为模拟图的变化，我们用在不同工作负载的迭代器动态改变的图评估我们的方法。我们在第一次工作量时使用图中75%的点，而后每次递增5%。最后，当工作负载工作5次，所有的点都得到使用。我们通过5次与其他划分算法的比较，评估HGR的运行时间。数据9(b)论证了我们方法关于图更新上的效率和稳健性。我们也设计实验评估工作负载在不同迭代器之间的状况。我们使用了5个工作负载：Semi-Clustering，Two-hop Friendship，Single Source Shortest Path,Breadth First Search 和Pagerank。各自分别记为迭代器1,2,3,4,5.我们用HGR和其他划分算法按顺序执行了这5个工作负载 。并将最后的工作负载Pagerank结果在论文中呈现出来。数据9(a)展现了我们方法的先进之处。

HGR的参数校正：正如我们在第3部分所提到的那样，我们使用参数k决定LAVS中点的尺寸。我们实施一系列实验研究k的选择。由于空间上的限制，我们仅对ECP为LAVS用Pagerank运行在Live-Journal数据集上的结果进行展示。如数据10所示。



a. b.

Figure 9: Job Execution Time on Live-Journal Dataset

我们发现当k=2或k=3是，HGR在大多数的工作负载中获得最佳表现。这是因为当k太小(k=1)时，LAVS尺寸太小，难以展现点之间的联系。当LAVS变大时(k>3)，LAVS包含太多不是有太强联系的点。因此我们使用k=3作为超步数，在实验中形成LAVS。

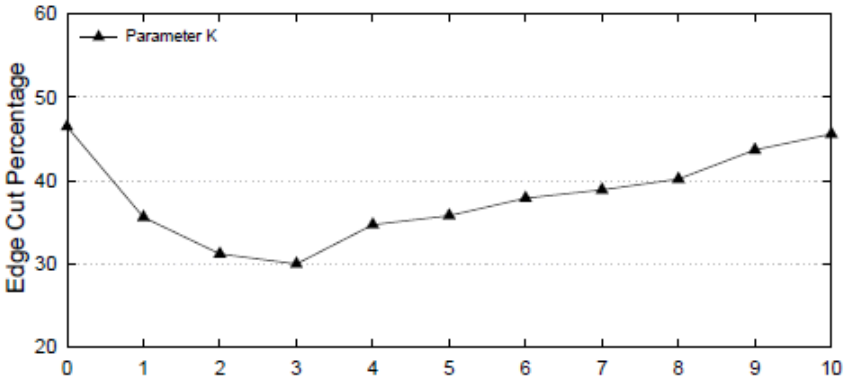


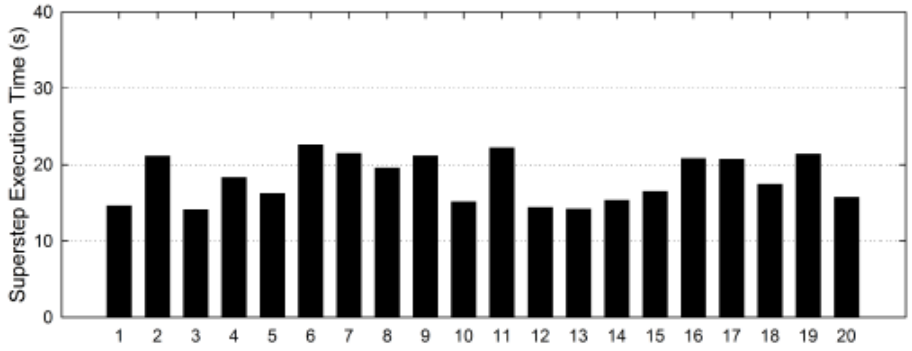
Figure 10: Parameter of LAVS Tuning

6.3 超步再划分的影响

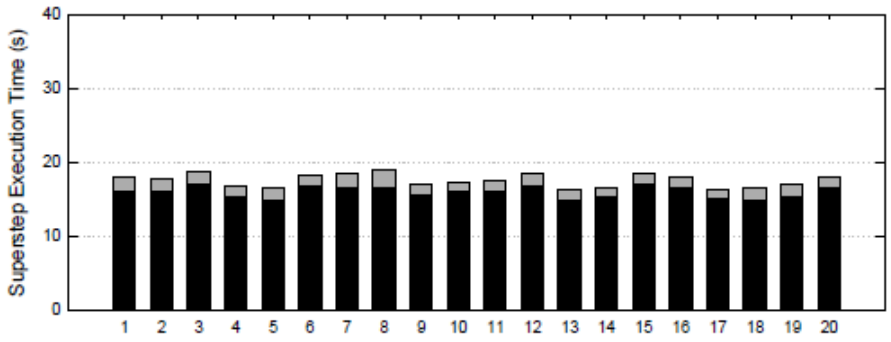
接下来，我们将呈现超步再划分(SR)。我们用紧密交互的Pagerank为代表惊醒评估。参数ϑ用来表示界定方程式12的开端的数字百分数，当被设置为2%时较其他值有更好的表现。对于超步再划分平衡每一次划分的运行时间，JET是一个比ECP更重要的标准。注意，SR的JET包含点再分配的时间。

为更好的理解超步再划分怎样平衡每一工作负载的运行时间，我们展示Pagerank工作负载下的第1个迭代器，运行在Live-Journal数据集上的一个样例结点。相关数据展示在数据11(a)中。第4超步相同结点的运行时间保存在数据11(b)中。我们观察经过3次点的改进，每个节点时间得到平衡，并且更短了。在我们实验中，JET包含再划分时间，在数据11(b)被标记为灰条。这些结果也确认了在LogGP系统中迁移和定位再分配点的效率(之高)。

接下来我们用不同数据集评估SR的影响，数据12(a)展示在不同图上运用LDG流划分作为初始划分，以10-iteration Pagerank工作量的JET的结果。注意，初始划分方法正交于超步再划分，我们接下来继续展示在不同初始划分结果上的结果。比较SR 和CatchWeb(在[21]中讨论，是一个动态图标工作负载平衡途径)。原始方法表示



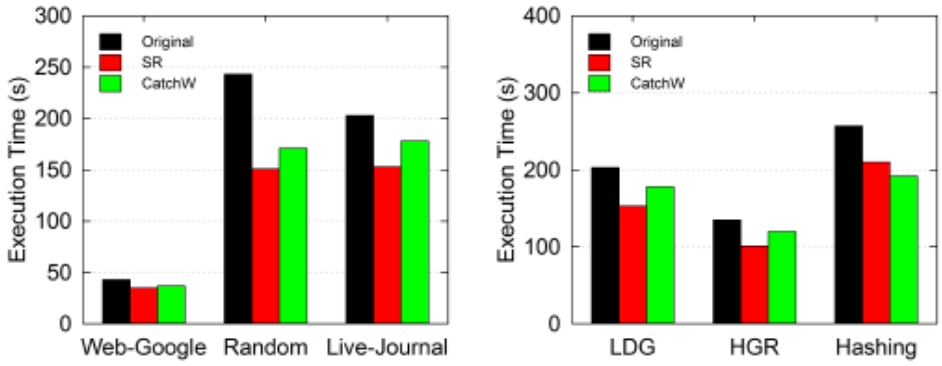
a. Execution Time of 1st Superstep



b. Execution Time of 4th Superstep

Figure 11: JET of One Sample Node in Different Supersteps

实验在超步中没有再划分。正如我们所看到的那样，当对比超步再划分缺位的方法，这些数据集上JET有明显的减少，减少为11.9%到27%。SR在所有3种数据集上都运行比CatchW快。



a. Performance of Pagerank b. Comparing with CatchW

Figure 12: Performance of Superstep Repartitioning

进一步评估不同初始划分对SR的影响。我们使用HGR，LDG和Hashing的方法作为初始划分结果。数据12(b)在3中再划分方法上的花销。我们将实验运行在Live-Journal数据集上进行10-iteration Pagerank的工作负载。如在数据12(b)所示，我们的再划分方法在HGR和LDG上表现良好。而在Hashing初始分区上没有太大进展。理由是，我们的算法集中注意力于平衡超步中每一结点的执行时间。哈希分区事实上更平衡，尽管运行起来要比LDG慢。对于LDG作为初始划分时，我们的方法由于CatchW，然而CatchW在Live-Journal数据集上又优于Hashing。CatchW初始设计用来从随机曲线划分开始，并重新分配点以减少通信开销。而我们的方法更多关注怎样根据从流初始划分的运行日志来平衡每个结点的运行时间。很显然，我们的方法在实际问题上效率更高。

这个实验表明了我们的超步再划分方法可以在执行期间平衡运行时间，并减少图处理工作的整个执行时间。

6.4 LogGP系统的优越性

从之前的实验中，我们可以看出超图再划分(HGR)和超步再划分(SR)分别在初始图划分和超图调整上取得不错的成果。在这个实验中，我们研究LogGP的整体表现，具备HGR和SR两方面的优势。

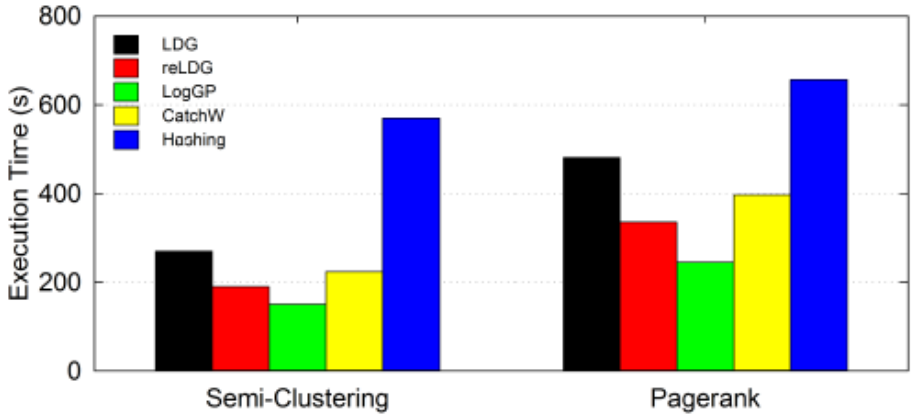
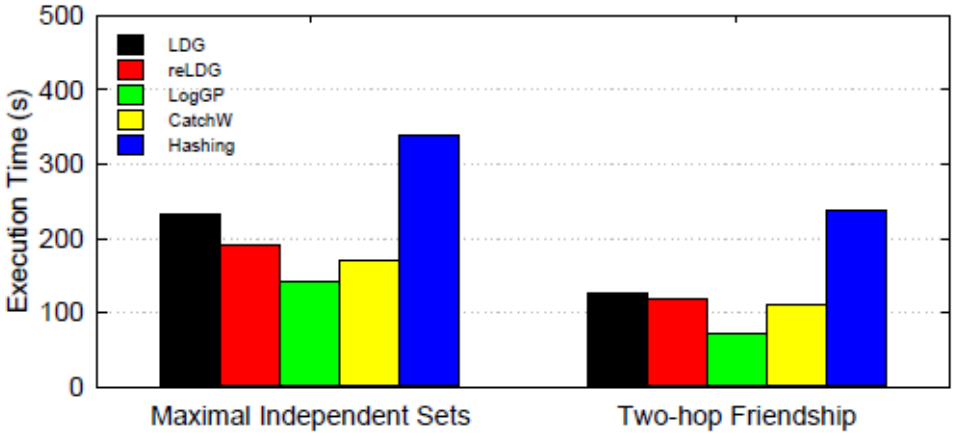
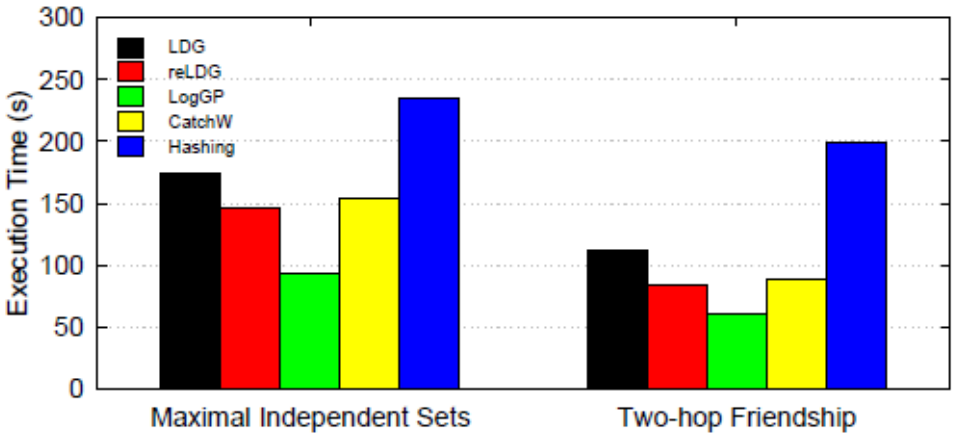


Figure 13: Overall Performance of logGP

数据13显示Twitter数据集上Pagerank 10-iteration和Semi-clustering工作负载下运行时间。我们比较以上所有的方法：LDG，reLDG和CatchW。正如我们所见，相较LDG，在Semi-clustering约有49.1%的减少，在Pagerank由39.2%的改进。LogGP表现优于CatchW和reLDG。这些结果确认了LogGP系统利用日志信息，较其他方法，可有效减少工作执行时间。



a. Wiki-Pedia



b. Wiki-Talk

Figure 14: Performance on Other Datasets

除了Twitter数据集，我们也在Wiki-Pedia和Wiki-Talk数据集上以工作量Maximal Independent和Two-hop Friendship 进行了评估。如数据14所示，LogGP在这些数据集和工作负载上表现均优于其他方法。这些结果也表明所提出的方法的一般性和它在不同领域的广泛应用(前景)。

接下来我们实施实验进一步研究整体的图片处理工作运行时间。我们在Live-Journal上运行Pagerank 10-iteration 工作负载。数据15展示了整个的运行时间(包括了图的加载时间(灰条)，划分时间(白条),和JET(黑条))。尽管引入少量的划分开销，我们的系统整体表现也还是最优秀的。

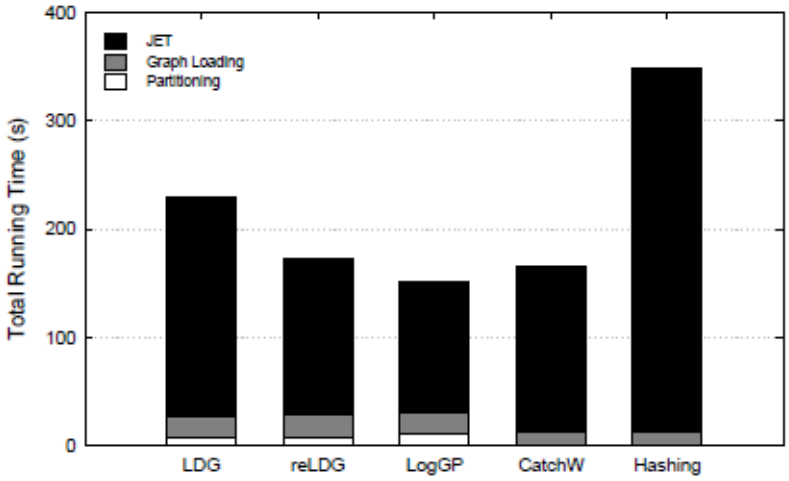
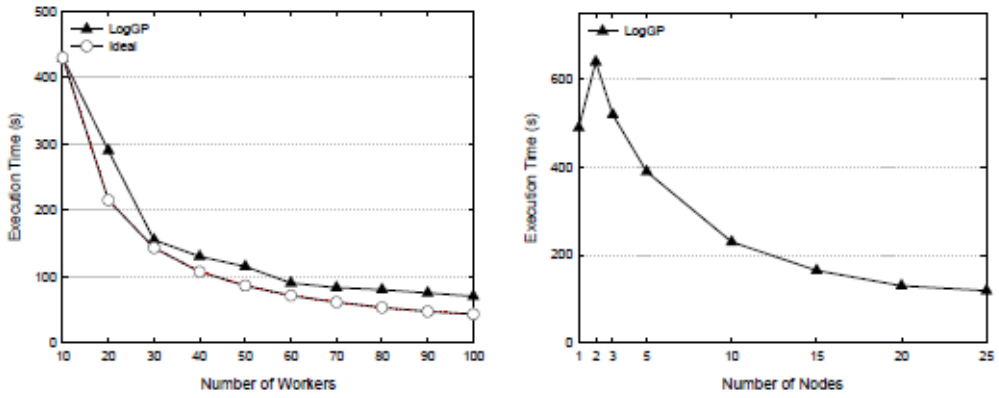


Figure 15: Study of Total Running Time



a. Different Workers b. Different Nodes

Figure 16: Performance of Scalability

可量测性研究：对相近的图形处理系统来说，可量测性是一项重要特征。我们进一步评估LogGP的可量测性，通过：

1) 固定数量的28个结点群和增加的工人(结点)；

2) 固定数量的100个工人(结点)和增加的结点数。

数据16(a)展示在Live-Journal上运行Pagerank 50-iteration的表现，工人个数由10个增加到100个。假定所得表现与工人个数线性相关，我们使用理想曲线刻画理想执行时间。。正如所预料的那样，随着工人个数增加，进步轻微减少。LogGP表现接近理想状态。这表明LogGP有一个很得体的可量测性。

我们也涉及了另一个实验，令结点个数不同，从1到25，同时工人的数量固

定为100。结果展示在16(b)中。如我们所料，在1个结点之上的表现要优于在2个或3个结点。因为有多个结点的群会在结点之间引入额外的通信开销。但是，很明显，如果我们继续增加结点数，LogGP表现会更好。如≥ 5时。因为群计算能力的获益超过通信方面的额外开销。

总结来说，基于我们的实验结果，得出结论：我们的LogGP充分利用历史日志信息优化划分，为大图处理系统提供一个有效的解决方案。

7.相关工作

这篇论文讨论的图划分问题涉及几个领域，如：图计算系统和图划分。

大规模图计算 ：为达到现在处理大尺度图的过高需求，许许多多方式和框架被提出，并受到欢迎。Pregel[17]和GraphLab[15]都采用一种以“顶点为中心”的计算模型，并同时在每个工作者结点运行由使用者定义的程序。Hama[2]和Giraph[1]都是开源项目，采用Pregel编程模型，并为HDFS作出相应的调整。在这些相似的图处理系统中，能将大图划分成几块平衡的次图，以便并行的工作者能够协调地处理它们，这点很重要。然而，大多数现行的系统多采用简单的哈希方式（hash method）。

图划分：图划分是一个组合优化问题，已被研究数十载。K-平衡图划分法旨在平衡顶点个数时最小化边切割数。尽管k-平衡图划分问题是一个NP-Complete 问题[10]，几个解决方案已经被提出用于应对该挑战。

Andreev et al.[6]提出了一个逼近（/近似）算法（approximation algorithm），用O(logn) 的近似度保证了多项式运行时间。另一个解决方案由Even et al.[9]提出，给LP一个基于可扩展指标的解决方案，也可以得到一个O(logn) 近似值。除了近似解决方案，Karypis et al.[14]提出了一种相似多级图划分算法，最小化每一级上的二分法。有一些探试实现，像METIS[13]，METIS[20]和Chaco[12]的类似版本被广泛使用在许多现有的系统内。尽管它们不能提供精确的性能保证，但它们的探试方法很有效。更多的启发式方法被总结在[3].
以上提到的方法都是离线的并且大体上都需要很长的处理时间。最近，Stanton和Kliot[23]提出一系列用探试的在线流划分方法（online streaming partitioning method）。Fennel[25]拓展了这项工作，他提出一个流划分框架。在这个框架下，可以组合其他探试的方法。Joel Nishimura and Johan Ugander [19]更深入地提出了Restreaming LDG 和 Restreaming Fennel，用最后的流划分结果，生成初始的图划分。Restreaming LDG 和 Restreaming Fennel开发如HGR相类似的策略。然而，HGR进一步使用日志活跃点集（the Log Active Vertex Set）得到顶点之间的内部关系。并将原图，最后一次流划分结果和日志活跃顶点集组合成一个超图。然后HGR再利用全局信息，使用一种新的超图流再划分算法（Hyper Graph Streaming Repartitioning algorithm）划分图。

除这些静态图划分技术外，[18]理论研究在没有重复加载或重复划分的系统开销，怎样调节图的结构变化。一些最近的工作[27,8]可以应对这些发生在图内的变化。但是，这些方法处理这些变化成本高昂。Shang et al.[21]研究几个图算法并提出简单有效的策略，当这种方法使用哈希划分作为初始输入的话，可以获得动态的

工作平衡。与我们的再划分方法SR相比，CatchW 尝试最小化整体的信息联系成本（total communication cost）。SR集中于解决系统的全部JET，更富有成效。除此以外，SR在执行过程中使用运行数据信息，并预测下一超步中每一结点的运行时间。在此预测的基础上，SR可以准确地移动合适的顶点。

8.结论

在这篇论文中，我们系统研究了在BSP模型上运行时间的不平衡。我们设计了一种新颖的基于日志的图划分系统，重复使用之前的和正在运行的日志信息改善划分。我们推出超图划分（Hyper Graph Partitioning）组合了图和历史划分结果生成超图（hyper graph）改善初始划分结果。当工作被执行时，我们用运行日志估计每个结点的运行时间并设计一种新颖的探试法，从偏离的结点中重新安排各顶点，以平衡运行的时间。样本和实验结果确认了我们新途径的改进。

有几个对我们将来工作很有前景的方向。首先，更深入的理论分析超图的流算法优于更好地成本预测。再者，其他重新安排顶点的探试方法（heuristic）也是个有意思的话题。

9. REFERENCES

[1] *Apache Giraph.* https://github.com/apache/giraph/.

[2] *Apache Hama.* http://hama.apache.org/.

[3] *Graph Archive Dataset.* http://staffweb.cms.gre.ac.uk/~wc06/partition/.

[4] *HDFS.* http://hadoop.apache.org/common/docs/current/hdfs/design.

[5] *Snap Dataset.* http://snap.stanford.edu/data/index.html.

[6] Konstantin Andreev and Harald Racke. Balanced graph partitioning. *Theor. Comp. Sys.*, 39(6).

[7] Rishan Chen, Mao Yang, Xuetian Weng, Byron Choi, Bingsheng He, and Xiaoming Li. Improving large graph processing on partitioned graphs in the cloud. In *Proc. of SOCC Conference*, pages 1–13, 2012.

[8] Raymond Cheng, Ji Hong, Aapo Kyrola, Youshan Miao, Xuetian Weng, Ming Wu, Fan Yang, Lidong Zhou, Feng Zhao, and Enhong Chen. Kineograph: taking the pulse of a fast-changing and connected world. In *Proc. of EuroSys*, pages 85–98, 2012.

[9] Guy Even, Joseph (Seffi) Naor, Satish Rao, and Baruch Schieber. Fast approximate graph partitioning algorithms. *SIAM Journal on Computing*, 28(6):2187–2214, 1999.

[10] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In *Proc. of STOC Conference*, pages 47–63, 1974.

[11] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: distributed graph-parallel computation on natural

graphs. In *Proc. of OSDI Conference*, pages 17–30, 2012.

[12] Bruce Hendrickson and Robert W Leland. A multi-level algorithm for partitioning graphs. *SC*, 95:28, 1995.

[13] George Karypis and Vipin Kumar. Multilevel graph partitioning schemes. In *Proc. of ICPP Conference*, pages 113–122, 1995.

[14] George Karypis and Vipin Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48(1):71–95, 1998.

[15] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proc. of VLDB Endow.*, 5(8):716–727, April 2012.

[16] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM journal on computing*, 15(4):1036–1053, 1986.

[17] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proc. of SIGMOD Conference*, pages 135–146, 2010.

[18] Vincenzo Nicosia, John Tang, Mirco Musolesi, Giovanni Russo, Cecilia Mascolo, and Vito Latora. Components in time-varying graphs. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(2):1–12, 2012.

[19] Joel Nishimura and Johan Ugander. Restreaming graph partitioning: Simple versatile algorithms for advanced balancing. In *Proc. of SIGKDD conference*, pages 1106–1114, 2013.

[20] Kirk Schloegel, George Karypis, and Vipin Kumar. Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience*, 14(3):219–240, 2002.

[21] Zechao Shang and Jeffrey Xu Yu. Catch the wind: Graph workload balancing on cloud. In *Proc. of ICDE Conference*, pages 553–564, 2013.

[22] Isabelle Stanton. Streaming balanced graph partitioning for random graphs. *arXiv preprint arXiv:1212.1121*, 2012.

[23] Isabelle Stanton and Gabriel Kliot. Streaming graph partitioning for large distributed graphs. In *Proc. of KDD Conference*, pages 1222–1230, 2012.

[24] Aubrey L Tatarowicz, Carlo Curino, Evan PC Jones, and Sam Madden. Lookup tables: Fine-grained

译者注：

1. 在 Pregel 中的计算是由一系列迭代组成的,被称为超步 s,文中未予以译出 。
2. heuristic algorithm 译为“探试算法”居多，也译为“启发式算法”。
3. Graph 译为“图”居多，也译为“图”，“表格”。