《Adaptive Partitioning for Large-Scale Graph Analytics in Geo-Distributed Data Centers》的复现及改进

谭浩彬

摘要

科技的进步使得大数据的应用得到了空前的发展，这些应用可以对用户数据进行分析，从而得知用户的兴趣爱好，精确投放内容，增加用户粘性。图处理算法正是为这样的分析服务的。而如今因为用户的分布广泛，为了提高用户使用应用时的体验，用户的数据是地理分布式的分布在不同地理位置的服务器中。为了分析这样一个地理分布式的图，将所有数据都同步到一个高性能服务器中，不仅花费大量的时间和迁移数据的成本，而且数据的更新不够及时，又浪费了其他地理分布式服务器的算力，不是一种很好的解决方法。因此，在地理分布式的图处理问题中，常常使用分布式的图处理算法。而在地理分布式的图处理中，如果不对图进行调整优化重新分区，会造成严重的通信时延和大量的通信费用。《Adaptive Partitioning for Large-Scale Graph Analytics in Geo-Distributed Data Centers》[1]一文中使用了强化学习的算法来优化图分割问题，取得了很好的结果。本次的课程论文将对其进行复现及改进。

关键词：图分割；图处理；地理分布式；强化学习

1 引言

请在此部分对选题背景，选题依据以及选题意义进行描述。

1.1 研究背景和意义

近年来，计算机性能高速发展，使得大数据的应用越来越熟练，例如常见的广告推荐、兴趣挖掘等。然而，这些算法的背后，依托着大量的用户数据来进行计算。比如常见的社交软件微博，大数据算法可以根据用户自身曾经发布、转发的微博，和关注者、粉丝发布、转发的微博，推断出用户的喜好，从而为用户量身定制内容的推送，加强用户的粘性。然而，这些用户的数据并不都是分布在同一个集群或者服务器之中的，而是地理分布式的。比如某些政府的官方微博，访问其的用户多是当地的群众，则该用户的数据大都生成并存储在当地或者靠近当地的服务器中，便于用户高速访问。当然，如果一位用户拥有几千万粉丝，几千万个节点同时访问同一个服务器非常浪费而且延时极高，所以，对于这些高访问的数据，一般会在各个地理分布式的服务器中各生成一个镜像节点。这样一来，用户靠访问最近的服务器的镜像节点，同样可以获取相同的内容，同时当主节点更新内容时，更新镜像节点是迅速且开销不大的。

地理分布式的数据提高了用户的体验，但是，如果需要对社交网络这个图进行数据分析的话，却是一个难点。其中最简单的方法就是将所有节点的数据迁移到一个服务器中进行图处理，但这种方法首先需要产生数据迁移的开销。这种开销是不可忽略的，因为是几千万甚至上亿的节点的数据量的输送，甚至还有数据传输时间的因素在里面，还有集群利用率低下的可能性。地理分布式图计算是一种新兴的计算模型，正好用于解决这样的问题。

地理分布式图计算被广泛应用于大数据分析等领域，包括流行的自然语言处理、推荐算法、信息检索等。应用了地理分布式图计算的模型后，每个地理分布式的服务器都参与了运算，且每个服务器之间的交流仅仅是图处理算法运行所需的中间变量（如一个浮点数），很好的提高了性能的使用率。同时，这样更是避免了需要大量搬运节点数据至其他服务器所产生的巨额开销。在地理分布式中的图计算模型中，首先会对一个图进行分割，将图的顶点及边分配到不同的地理分布式服务器中，同时生成镜像，用以保存图节点在运行图处理算法时需要的邻居关系。目前常见的图分割模型有点切割[2]、边切割[3]和混合切割[4]。其中混合切割结合了点切割和边切割的方法，根据节点的度数判断用两种切割方法的哪一种，在应对幂律分布[4]的自然图中，能够有效较低镜像的生成，产生较低的通信时延和通信费用。基于此，进行图分割后，每个地理分布式服务器中都含有图的一部分且能够保持每部分节点与其他节点的有效通信。接着，每个地理分布式服务器中就可以运行实际需要的图处理算法，比如PageRank[5]，单源最短路径等。

但是，在地理分布式的服务器中，每个服务器由于距离远近、维护成本等因素，导致每个服务器的网络性能参差不齐，具体表现为每个服务器的上传/下载带宽不一致、流量价格不一致等。如果直接在这样的地理分布式环境中直接进行图处理，会由于网络参数的不一致而带来性能瓶颈，造成图处理算法的效率低下。目前的一些图分割的优化算法[6][7][8][9]大都基于启发式的算法，且将所有分割区域当成是无差异的，并不适合我们的问题。同时，由于图的规模、地理分布式服务器的网络差异、优化目标等种种因素，每个节点的位置及其镜像的位置都会对地理分布式图处理性能造成影响，所以地理分布式图分区问题的解空间是非常大的，达到 $O(M^N)$，其中 N 是节点数量，M 是可迁移的地理分布式服务器数量。在面对这一问题中，暴力法实不可取的，而传统的启发式算法优化程度有限且效率低下，更是需要充分的前置知识根据不同的环境去设计。目前，强化学习最近常被用在进行复杂决策的场景中，且对复杂的动态环境中自主学习找到合适的长期回报。而地理分布式图分区的问题也与决策息息相关，

1.2 国内外研究现状

地理分布式图分区问题的核心在于如何将一个图分割成适合该地理分布式环境的部分，即图分割模型。目前主流的图分割模型有点切割[2]、边切割[3]和混合切割[4]。常见的分布式图计算模型中有Pregel[10]和 GAS[2]两种。Pregel 应用的是边切割模型。然而，这种切割以边为中心进行切割，虽然保持了每个节点都有边的信息，但是在对于幂律分布[4]的图而言，由于边数的不确定，可能会导致某些部分的工作负载高，导致效率低下。GAS 应用的是点切割模型。这种切割以节点为中心，在多个切割部分产生同一节点的镜像节点，可以将负载较好的分配至不同的部分。但是，这种切割会导致节点与镜像的通信带来的跨区域通信量增多。混合切割综合应用了点切割和边切割两种方法，以节点的度数判断使用两种中的哪一种，很好的缓解了两种单一模型的缺点，带来通信效率提高以及负载均衡。

基于这三种基本的图分割模型，现有的大多数图分割优化方法主要是决策每个节点的位置，再应用上述的三种基本图分割模型，来优化整体的图处理性能。部分算法[2][3][4][11]把负载平衡和通信最小化作为优化目标，这些方法在具有同质网络带宽的传统系统中可以带来良好的图处理性能，而在网络参数不尽相同的地理分布式的服务器下，并不能够如意。少有的一部分算法[6][7][8][9]能够在异质性的环境下完成图分割，但这类算法本质上都是一些启发式的算法，而启发式算法往往需要一些前置知识

地区	上传速率 (MB/s)	下载速率 (MB/s)	上传价格 ($ /GB)
US East	100	500	0.02
AP Singapore	100	500	0.09
AP Sydney	50	100	0.09

表 1: Amazon EC2 t2.small 实例从对应地区到互联网的上传/下载带宽

和经验去设计，且搜索空间大而效率低下，导致最终得不到良好的优化效果。Mayer C 等人[7]提出了 GrapH，一种自适应的流式分区方法，这种方法考虑了顶点流量、网络价格之间的异构性，目标是减少图计算过程中的通信开销。然而，GrapH 没有考虑网络带宽之间的异构，因此可能会导致非常大的数据传输时间开销。Zhou A C 等人[6]提出了一种在面对地理分布式环境下的图的重分区算法，考虑了网络带宽、网络价格之间的异构，但是这种算法采用服务器互换的思路，仍然是一种启发式的算法，且没有在节点的层次上进行分区，虽然能够获得较好的结果，但是性能提升不够大，容易陷入局部最优解中。Mofrad M H 等人[12]提出了使用强化学习的算法来进行图分区，它基于有模型类型的强化学习算法，不断的进行策略迭代，目标是得到一个负载平衡且本地性能好的图分区。这种方法同样没有考虑到网络中价格以及带宽的差异性，不能直接适用于本文的问题。强化学习最近常被用在进行复杂决策的场景中，且对复杂的动态环境中自主学习找到合适的长期回报，所以本文将强化学习算法进行重新设计，使得在地理分布式的环境下分区后能够提高性能。

2 背景

2.1 地理分布式图处理

现有的大多图数据，例如社交网络，都被分布在地理分布式的服务器中。而对这样的图进行分析的话，最简单的办法就是把所有的数据都集中到一个服务器或者集群中。这样的做法不但需要大量的带宽，而且由于地理分布式的服务器距离的限制，传输还需要很长的延时，导致需要花费大量的流量开销、时间开销以及得不到最新的数据更新。更重要的是，不同国家的隐私政策不完全相同，可能不允许数据的随意迁移[12]。同时，对于像 twitter 这一类在全球都有用户的庞大的应用的图数据，存储数据的服务器分布在全球各地，网络差异性更为严重。所以，现有的方法大都是在地理分布式的服务器上进行分布式的图处理。分布式的图处理算法存在以下优点：首先只需要传输图处理时产生的跨服务器的通信，大大降低流量；其次保证了数据的安全，因为图节点的数据并不需要迁移，保证不会触碰到不同国家的各不相同的隐私政策；还可以在多个服务器并行的运行图处理算法，提高算力。但是，对于地理分布式的服务器，存在以下几个需要注意的地方。

- 地理分布式的服务器除了网络参数外，大都使用相同的运行配置。
- 尽管是同一个实例，但是由于地理位置的不同，网络参数不尽相同。表格 1给出了 Amazon EC2 t2.small 实例在不同地理环境下的网络参数。
- 经过查询后发现，同一个地区的集群内的服务器之间通信，使用的是局域网，不会收取网络的流量费。同时由于是在同一个地区，传输延迟对比与跨地区的服务器通信而言基本可以忽略。

综上所述，在地理分布式的服务器的图处理应用上，其瓶颈在于网络的通信上。由于地理分布式的服务器一般采用同样配置的实例，且同一地区的局域网开销要比跨地区的广域网要小很多，实验中

本文并不考虑服务器自身计算时产生的时间开销和局域网通信带来的时延、价格开销.

2.2 图计算模型

为了分布式地分析图，需要用到相关的分布式图计算模型。目前的分布式图计算框架基本上都遵循 BSP（Bulk Synchronous Parallell）计算模式[13]，它将计算分成一系列的超步（superstep）的迭代。从纵向上看，它是一个串行模型，而从横向上看，它是一个并行的模型，每两个超步之间设置一个栅栏（barrier），即整体同步点，确定所有并行的计算都完成后再启动下一轮超步。常见的两个框架分别为 Pregel 和 GAS。Pregel 建立在 BSP 的编程模型上，用户在编程时主要围绕以节点为中心，每对节点之间的通信采用纯信息发送的形式。GAS 模型建立在异步分布式共享内存的编程模型，每个节点可直接对相邻节点（或镜像）进行直接的修改，再由同步的阶段进行节点与镜像的通信同步。GAS 为 Gather、Apply、Scatter 这三个阶段的缩写。其中，Gather 阶段，镜像节点负责收集邻顶的消息后进行处理，这种处理可以是求和、求极值，由算法进行决定。Apply 阶段，镜像节点将收集到的消息发送至主节点，主节点对所有收集的镜像节点消息与自己本地收集的消息进行处理，更新自身的状态。Scatter 阶段，主节点将更新后的自身属性（值）发送给镜像节点进行更新，确保读取镜像节点的属性（值）时是最新的。经过 Gather、Apply、Scatter 这三个阶段后，就完成了图计算一次迭代的通信，且保证了数据的正确性。本文后续实验将使用 GAS 模型。

2.3 图分割模型

为了在分布式中更好的进行图处理，需要把一个图进行分割，而如何将一个图分割成几部分后，仍然建立可靠的联系，是图分割模型来解决的。目前常见的图分割模型有点切割、边切割和混合切割。

- 点切割是以节点为中心，将节点以及其入边（或者出边）放置同一个分割部分，然后补充邻节点的镜像（如果主节点不在同一部分的话）。
- 边切割则是以边为中心，将边分布在不同的部分后，补充节点的镜像。
- 混合切割则使用了两种方式进行结合，引入系数 λ，对于入边度数（或者出边度数）大于等于 λ 的节点，采用边切割；否则使用点切割。入边度数（或者出边度数）大于等于 λ 的节点称为 High Degree Node，其余称为 Low Degree Node。

混合切割的做法更适用于社交图等符合幂律分布[2]的图，本文将使用混合分割作为实验部分图分割的模型，应用到 GAS 图计算模型上。所谓幂律分布，就是指图中大部分的节点的度都是较小的，而只有小部分的节点的度数较大的。如社交网络中，名人的关注度很高，但是名人的数量却很少；而大部分用户都是普通人，对于普通人的关注度往往不高。在这种幂律分布的图中，如果仅使用边切割，则会造成高的通信成本和负载不平衡；如果仅使用点切割，又会造成大量的复制镜像的产生，浪费通信资源。因此，合理的调整 λ 使用混合切割，会很好的降低镜像的复制率，从而减低通信成本。

图 1展示了一个图使用 3 种不同分割方式带来的效果，当 $\lambda = 2$ 时，对节点 1 采用点切割，而对其他采用边切割。可以发现，混合切割考虑到了顶点的自身属性不同（指入度），使得镜像的数量大幅减少。获得较低的节点镜像复制率在地理分布式服务器中很重要，因为多量的镜像与主节点体之间的跨服务器数据通信会导致地理分布式图处理的性能变差、成本变高。本文将使用混合切割的图分割方法进行实验。

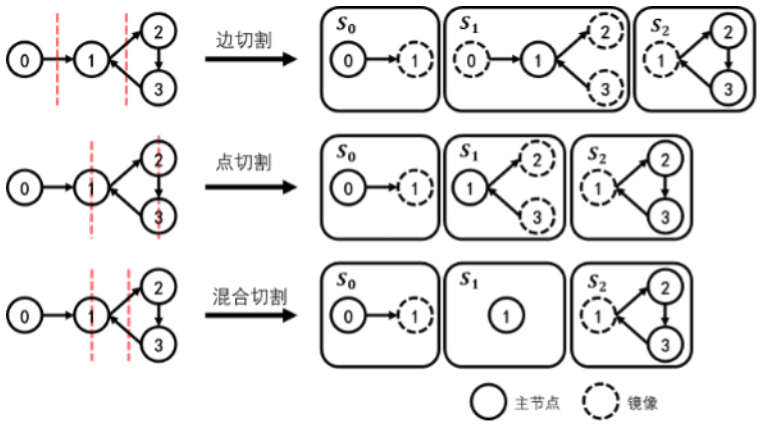

图 1: 三种不同的图分割模型

2.4 图处理算法

图处理算法广泛的用于分析图中每个节点的各个特征，可以用来排名、重要性分析等，以此提取出一个图的特征情况。常见的图处理算法有 PageRank[5]、SSSP（单源最短路径算法）等，本文将介绍并使用 PageRank 算法。PageRank，又称网页排名、谷歌左侧排名，是一种由搜索引擎根据网页之间相互的超链接计算的技术，而作为网页排名的要素之一，以 Google 公司创办人拉里 • 佩奇（Larry Page）之姓来命名，同时也是目前应用最为广泛的图处理算法。Google 用它来体现网页的相关性和重要性，在搜索引擎优化操作中是经常被用来评估网页优化的成效因素之一，是常见的图处理算法之一。算法 1 展示了 PageRank 算法的步骤，在一个迭代中，每个节点分别收集它们入边相邻节点的 Rank 值，进行求和后更新自身的 Rank 值，并将更新后的 Rank 值除以出边度数后发送至它们的出边邻点，提供给下一个迭代时出边相邻节点收集所用。其中，K 是算法的迭代次数，$PR(u)$ 表示节点 u 的 Rank 值，$N_{in}(u)$ 表示节点 u 的所有入边邻点，$N_{out}(u)$ 表示节点 u 的所有出边邻点，$M(u)$ 表示最新的 Rank 值除以出边度数后得到的平均值。

Procedure 1 PageRank

Input: G: the graph dataset; K : max number of superstep
for *k=0* to K **do**
  for *vertex u* in G **do**
    $PR(u) = 0.15 + 0.85 \sum_{v \in N_{in}(u)} M(v)$
    $M(u) = PR(u)/|N_{out}(u)|$
    Send $M(u)$ to all $N_{out}(u)$
  end
end

2.5 学习自动机

学习自动机（Learning Automaton）是机器学习中的一类算法，它通过不断的与环境进行交互，以一定的概率执行某些动作，然后取得奖励或惩罚，通过奖励或惩罚来调整对应执行的动作的概率，以此来完成概率收敛到最佳动作。$\{A(n), P(n), R(n), T\}$ 可以用来表示学习自动机的 4 个阶段：n 表示当前的迭代数；$\{A(n) = \{a_0(n), a_1(n), ..., a_m(n),$ 表示共有 m 个动作空间可供选择执行；$P(n) = \{p_0(n), p_1(n), ..., p_m(n),\}$ 分别表示每个动作执行的概率分布，自动机执行动作时将会按照该概率分布进行随机选择；$R(n) = \{r_0(n), r_1(n), ..., r_m(n),\}$ 表示执行每个动作后分别取得的反馈，其中 $r_i(n) \in \{0, 1\}$，1 表示这个动作是好的，0 表示不好；T 是更新概率的函数，用于 $P(n+1) = T(A(n), P(n), R(n))$：

假设在 n 迭代中，执行动作 $a_i(n)$ 后得到的 $r_i(n)=1$，则

$$P(n+1)=\begin{cases} p_j(n)+\alpha(1-p_j(n)), i=j \\ p_j(n)(1-\alpha), i\neq j \end{cases} \tag{1}$$

经过多次实验，作者发现根据负反馈（$r=0$) 来调节概率会导致过慢的收敛，因此并未加入。

3 本文方法

3.1 问题描述

对于一个图 $G=V,E$，V 表示所有节点的集合，E 表示所有边的集合。其中，节点起始分布在 |M| 个地理分布式服务器中，每个顶点所在的服务器为 $L(u)=m_i, u\in V, m_i\in M$。当节点的位置确定后，首先按照混合切割的方法将边分布在不同的服务器上，最后将需要的镜像节点进行复制到对应的服务器上。如图 2所示，图共有 5 个节点和 3 个地理分布式服务器。假设 $\lambda=4$，则 0 号节点为 High Degree Node，其余为 Low Degree Node。首先进行节点的分布，图中的初始位置分布为节点编号 hash 服务器个数；然后，对边进行分配，所有的 Low Degree Node 把各自的入边移动至各自服务器中，而 High Degree Node 的入边以邻节点出边的形式移动到邻节点所在的服务器中；最后，补充边所需要的镜像节点，完成混合切割。

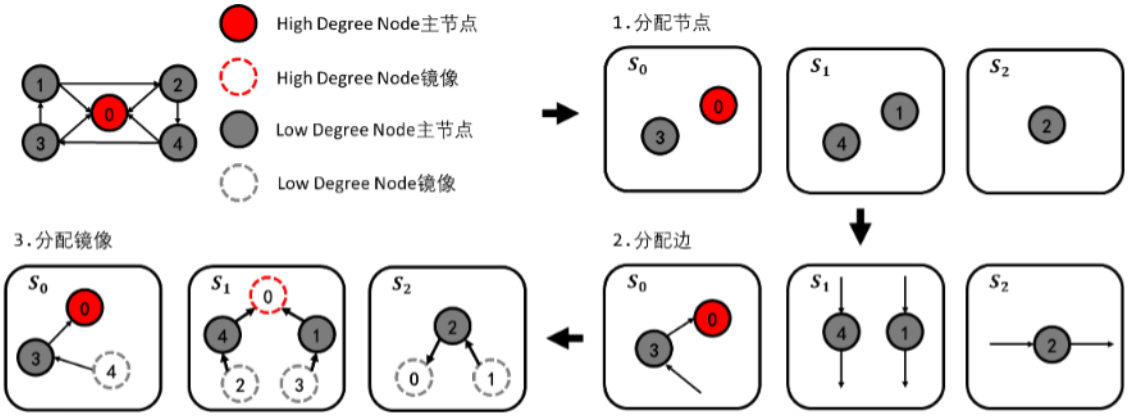

图 2: 混合切割样例

在 GAS 通信模型中，由于 Gather 阶段属于本地消息的收集阶段，不涉及到跨服务器通信，所以在计算一次迭代的开销时，可以忽略掉 Gather 的开销。对于 PageRank 算法，本文假设消息沿着出边发送。在 Apply 阶段中，主节点需要接收所有入度不为 0 的镜像节点发来的 Gather 阶段收集到的消息。Apply 阶段所花费的通信时延包括每个服务器镜像节点上传和主节点下载两部分的时间取最大值，即

$$T_A^r=\max\left(\frac{\sum_v I_v In_v W_A\left(r,L\left(g\left(v\right)\right)\right)}{Upload_r},\frac{\sum_v\left(1-I_v\right)H_v\sum_{m\in R_v}In_m W_A\left(L\left(m\right),r\right)}{Download_r}\right) \tag{2}$$

其中，r 是服务器的编号,T_A^r 代表 r 服务器在 Apply 阶段的通信时间消耗；I_v 表示在 v 节点为镜像，是则返回 1，反之返回 0；In_v 返回 1 当节点 v 的入边度数 $\neq 0$，否则返回 0；$W_A\left(r_1,r_2\right)$ 表示算法在 Apply 阶段一个节点从服务器 r_1 发往 r_2 的通信数据量，该值由对应的图处理算法决定；$g\left(v\right)$ 表示节点 v 的主节点；H_v 返回 1 当 v 节点为 High Degree Node；R_v 返回节点 v 的所有镜像;$Upload_r$ 和 $Download_r$ 分别表示服务器 r 的上传和下载带宽。对应的，在 Scatter 阶段，通信时延包括 mirror 接收更新的数据和 master 发送更新的数据所产生的时间消耗，即

$$T_S^r=\max\left(\frac{\sum_v\left(1-I_v\right)\sum_{m\in R_v}Out_m W_S\left(r,\ L\left(m\right)\right)}{Upload_r},\frac{\sum_v I_v Out_v W_S\left(L\left(g\left(v\right)\right),r\right)}{Download_r}\right) \tag{3}$$

其中，r 是服务器的编号,T_S^r 代表 r 服务器在 Scatter 阶段的通信时间消耗；Out_v 返回 1 当节点 v 的出边度数 $\neq 0$，否则返回 0；$W_S(r_1, r_2)$ 表示算法在 Scatter 阶段一个节点从服务器 r_1 发往 r_2 的通信数据量，该值由对应的图处理算法决定。综上，一次 GAS 迭代的总通信时延为

$$T = T_A + T_S = \max_r T_A^r + \max_r T_S^r \tag{4}$$

经过调查，在地理分布式的服务器中，一般只有服务器上传的流量才会进行收费，所以，在问题模型中，本文不考虑下载所消耗的流量带来的费用开销，仅仅考虑上传所消耗的流量所带来的开销。综上，一次 GAS 迭代的总流量费用开销为

$$Comm\ cost = \sum_v \sum_{m \in R_v} H_v In(m) W_A(L(m), L(v)) P(L(m)) + (1 - H_v) Out(m) W_S(L(v), L(m)) P(L(v)) \tag{5}$$

其中，P(r) 表示 r 服务器的上传的单位价格。

对于移动节点，当然不可能毫无限制的进行移动。首先，每个节点都会有自己自身的数据部分，例如社交网络中，每个用户的个人信息以及其发布的照片、文章等资源，这些都被分布式地存储在地理分布式的服务器中。为了简化问题本文假设这部分数据是完全相同的大小 D。当进行节点的迁移时，这部分大小为 D 的数据要求被完全的移动至另一台地理分布式的服务器中，由于上述提及下载的流量开销可以忽视，且本文主要关注进行图处理时的性能情况，所以本文在问题上只将迁移节点时产生的上传数据的流量价格开销纳入考虑的范围。易知，假设服务器上所有节点都进行迁移时，产生最昂贵的服务器为 r（即服务器的主节点数 $\times D\times$ 上传价格，最昂贵的那一个服务器），则将除了 r 以外的服务器上的所有节点都迁移到 r 服务器上，所有的节点都在一个服务器上，图处理的通信时延和通信费用开销都是最优解（都是 0）且迁移成本也是相对最少的。本文把这样的迁移方案的迁移开销记作 B。为了符合现实的情况，不将所有的节点移动至同一服务器上，本文引入一个变量 $B_{rate} \in [0, 1]$。在模型上，本文希望只产生不超过预算 $budget = B \times B_{rate}$ 的迁移成本来提高图处理应用时的性能。这样的考虑原因有：在现实中，大规模的移动数据是及其缓慢且大量开销的，尤其是在地理分布式服务器的环境中，本文希望通过只移动少量的节点来尽量优化图处理应用时的性能就好；其次，为什么用 $B \times B_{rate}$ 而不是节点数量的百分比，也是因为考虑到地理分布式服务器的环境，如果简简单单的使用节点的数量进行约束，无法有效的考虑到地理分布式服务器的差异，且使用 $B \times B_{rate}$ 直接对价格进行约束，更符合人们对预算的观察与决策。综上，问题可以描述为：

$$\begin{cases} min(T) \\ min(Comm\ cost) \\ move\ cost \leqslant budget \end{cases} \tag{6}$$

3.2 奖励函数

本文使用的反馈函数如下：

$$\theta = \frac{iter}{MAX\_ITER} \tag{7}$$

$$score_{base} = \frac{T_{old} - T_{new}}{T_{old}} + \frac{Comm\ cost_{\ old} - Comm\ cost_{new}}{Comm\ cost_{old}} \tag{8}$$

$$score = \begin{cases} scorebase, & move\ cost_{new} < budget \\ (1-\theta)scorebase + \theta\dfrac{move\ cost_{old} - move\ cost_{new}}{movecost_{old}}, & else \end{cases} \tag{9}$$

其中，iter 是当前迭代的次数；MAX_ITER 是学习自动机的最大迭代次数；T_{old} 是进行动作前图处理应用时的一次迭代的通信时延，T_{new} 是进行动作后图处理应用时的一次迭代的最新通信时延；$move\ cost_{new}$ 是完成迁移后的迁移成本，$move\ cost_{old}$ 是完成迁移前的迁移成本；$Comm\ cost_{old}$ 是进行动作前图处理应用时的一次迭代的通信费用，$Comm\ cost_{new}$ 是进行动作后图处理应用时的一次迭代的最新通信费用； score 公式的意义为：当迁移所产生的迁移成本小于预算时，这时节点可以进行随意的迁移，迁移的好坏由图处理应用通信时延的优化百分比和图处理应用通信费用开销的优化百分比求和组成。这里允许时间和价钱不同时优化，但要求其中一方的优化至少要大于另一方可能产生的负优化。当迁移所产生的迁移成本大于预算时，这时需要将成本开销放入优化的范围内，也就是说，将节点迁移回最初的位置时也是一种优化，使迁移成本重新满足预算内。但是，一味地将成本限制也是不好的。在强化学习算法中，智能体需要不断的通过与环境交互来学习并且适应最先的环境。尤其是在多智能体中，一个智能体的动作带来的结果改变了其他的智能体的环境变换。如果一开始就限制 move cost，则很可能将结果限制在了一个局部解中。所以，在迁移所产生的价钱开销大于预算时，会有由于超过了预算而带来的惩罚。但这种惩罚随着迭代次数的递进而不断增强，这样的意义为：在训练的前期，本文希望智能体多多去探索更多的解空间，通过 θ 去让智能体更多的注意在图处理性能的优化效果上；在训练后期，本文希望智能体更多的注重实际问题上的限制，尽可能的找到符合预算且对于性能有优化的解。

3.3 撤销与采样

在本文的算法中，经过动作执行阶段后，环境会返回一个与动作对应的反馈 r。r 的正负在一定程度上表现了动作执行后对环境的改变的好坏。我们可以想到贪心算法，既然对于环境的动作又好也有坏，那么可以将不好的动作舍弃掉，只执行好的动作。这样一来，一来能够保证环境的变化一定是往好的方向的，二来因为撤销了动作所以能够减少解空间的探索，使智能体更快收敛。

同时为了减少训练开销，本文中使用了随机采样智能体进行训练及动作执行的方法，通过减少训练的智能体的个数来减少时间开销。具体流程如图 3所示。

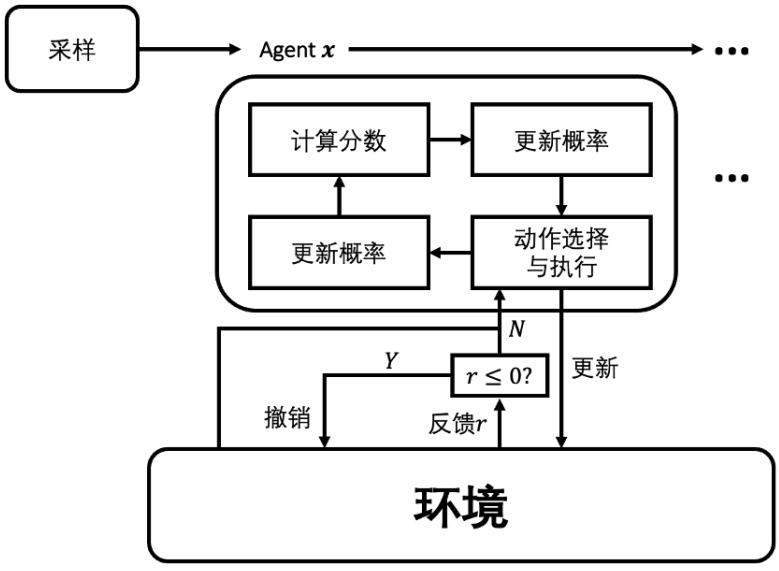

图 3: 算法流程

4 复现细节

与开源代码相比，我同样使用 C++ 进行编程，我的复现改进部分主要包含编程部分和优化采样部分。

4.1 编程优化

与开源代码相比，我的代码中：

1. 开源代码中在图结构进行保存的时候使用的数据结构是 List 链表。虽然链表在增、删上有时间优势，但这部分优势只有在构建图的时候有效。然而，计算系统的开销时候重点在强化学习的时间开销上，并不在图的构建上，反而 List 的遍历需要更多的访存时间，消耗更大。因此，在复现时我选择使用 Vector 可变数组来保存图数据。
2. 开源代码进行顶点映射时使用的是 C++ 中的 map，而 map 的实现是红黑树，为了加快算法，我改为了基于哈希表的 unordered_map。
3. 开源代码的并行部分使用的是 C++ 中的 pthead，通过手动的同步来完成。我使用了 C++ 的 openMP 库来加快程序的执行。
4. 开源代码中的编译部分（MAKEFILE 文件中）只使用了-O 生成程序，我使用了-O3 来进行变量的优化，加快程序的执行。

4.2 采样优化

开源代码中使用的是随机采样的方法。首先对问题进行分析，图处理算法时的性能优化方向有时间和价格两种，这两种都是由于主节点与镜像节点进行信息交换而产生的，所以，要想提高图处理算法时的性能，减少镜像的生成是一个重要的突破口。然而我们可以逆向思考，如果对节点进行训练，一定会减少镜像的生成吗？如图 4a 所示，S_0, S_1, S_2 中为实际的节点及其镜像的分布情况，红色节点的镜像节点在所有的服务器中都存在镜像节点，那么无论进行什么动作的选择，最后最好的优化都会是由于地理分布式的服务器的网络参数不同所造成的。更极端的是，假如地理分布式的服务器的网络参数相差并不大，甚至是没有差别，那么无论选择那个服务器作为主节点，都是没有任何优化的，只能把其中一个镜像的所有邻居节点都搬到任意其他的一个服务器才能减少一个镜像的生成。

在论文中，作者使用的是基于图顶点的度数从小到大来进行动态采样。这样的采样是基于图结构的，而图结构在整个过程中是一成不变的。而一种更好的方法应该是根据与图的动态信息——图顶点的镜像数量来决定。由此以来，采样的依据由静态的数据变成了动态的数据。但是，这也衍生出其他问题：

1. 随着训练的进行，顶点的镜像数量在动态变化，因为动态变化而导致的额外开销怎么处理？
2. 训练镜像数量高还是低的顶点对应的智能体收益更高？

4.3 实验环境搭建

- 图处理算法

  实验中，本文选用 PageRank 算法作为需要优化的图处理算法。在算法中，本文假设 PageRank 传输的单位数据量为一个 double 值，即 $W_A = W_s = 8Byte$。本文以此作为环境去优化 PageRank 的执行性能。

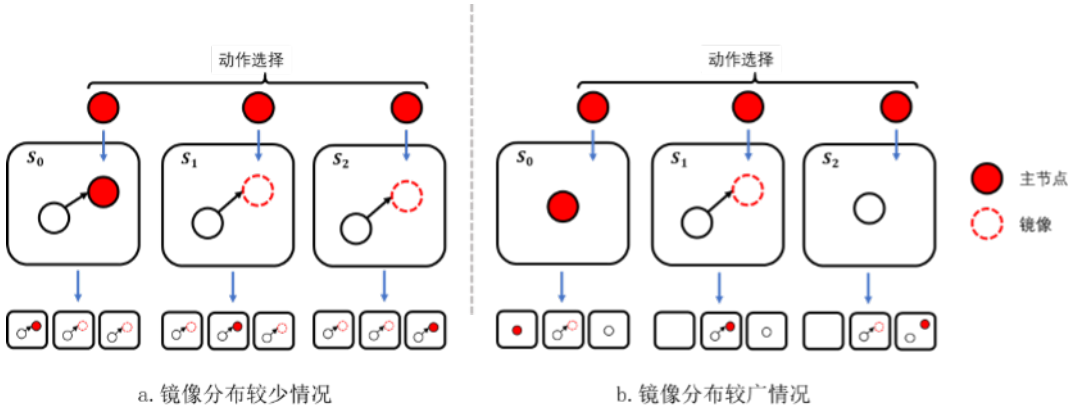

图 4: 两种镜像分布不同情况的动作选择产生的结果

- 网络环境

  在实验中，本文使用亚马逊 cc2.8xlarge EC2 实例在八个不同地方的实际网络参数，创建了八个服务器，进行模拟实验，如表 2所示

地区	上传速率 (MB/s)	下载速率 (MB/s)	上传价格 ($ /GB)
US East	100	500	0.02
US West Oregon	100	500	0.16
US West North California	100	500	0.02
EU Ireland	500	100	0.02
Asia Pacific Singapore	100	500	0.02
Asia Pacific Tokyo	100	500	0.09
Asia Pacific Sydney	50	100	0.09
South America	100	500	0.14

表 2: 实验所使用的地理分布式服务器参数信息

- 服务器配置

  程序在服务器中运行，服务器配置如表 3所示。

CPU	Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.50GHz × 2，共 48 核
内存	64GB DDR4 * 16，共 1TB
系统	Linux version 4.15.0-20-generic (buildd@lgw01-amd64-039)
C++	gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1 18.04)

表 3: 实验所使用的运行服务器参数

5 实验结果分析

实验结果如图 5图 6所示。其中 sort1 表示只在算法执行前进行一次智能体镜像数量的排序，sortt 表示每次迭代都重新进行一次智能体镜像数量的排序；H2L 表示排序的结果是从镜像数量高到镜像数量低进行排序，L2H 表示排序结果是从镜像数量低到高进行排序。RLcut(origin) 表示开源代码执行结果，RLcut 表示经过编程优化后的执行结果。

可以看出，经过编程优化的代码结果要好于开源代码，这是因为经过优化后，代码执行速度更快，能够训练的智能体数量更多。sort1 结果要好于 sortt，因为排序的耗时无法省略，过多的排序虽然能够及时的调整训练的节点，但是额外的耗时又反过来缩小了训练的时间，所以只排序一次即可。H2L 的结果要好于 L2H，这是因为镜像数量较少的智能体（如只包含 0/1 个镜像）本身优化的空间就比较小了。

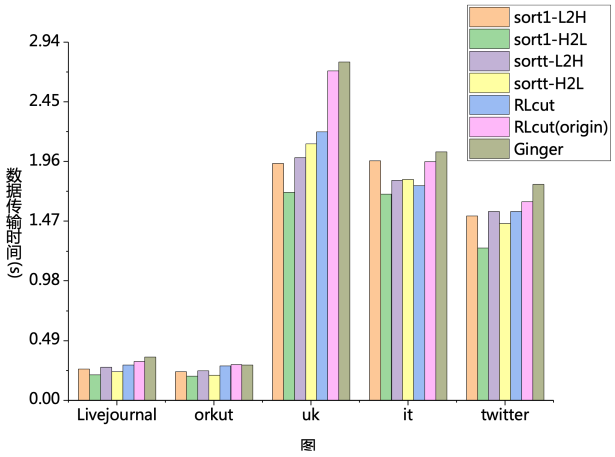

图 5: 传输时间优化对比

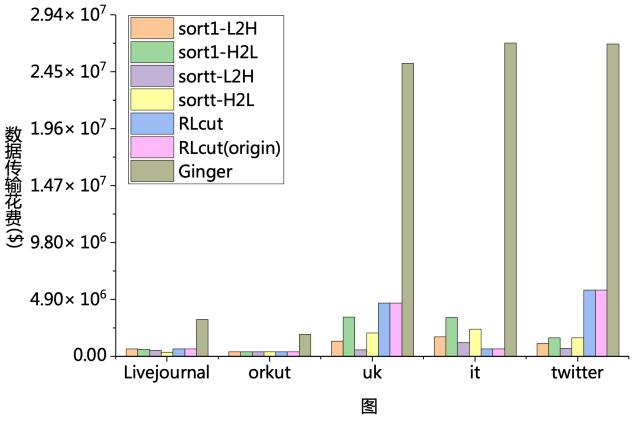

图 6: 传输开销优化对比

6 总结与展望

本文介绍了如何使用强化学习算法解决地理分布式图分割问题，最后整理了复现的要点和改进的地方，虽然很小但足够有用。本文不足之处是没有完全复现论文中的其他两种图分析算法（SSSP 和 subgraph）。最后，本次复现中想到了很多实践上的问题，对日后的工作有较大的帮助。

参考文献

[1] ZHOU A C, LUO J, QIU R, et al. Adaptive Partitioning for Large-Scale Graph Analytics in Geo-Distributed Data Centers[C]//2022 IEEE 38th International Conference on Data Engineering (ICDE). 2022: 2818-2830.

[2] GONZALEZ J E, LOW Y, GU H, et al. {PowerGraph}: Distributed {Graph-Parallel} Computation on Natural Graphs[C]//10th USENIX symposium on operating systems design and implementation (OSDI 12). 2012: 17-30.

[3] LOW Y, GONZALEZ J E, KYROLA A, et al. Graphlab: A new framework for parallel machine learning [J]. arXiv preprint arXiv:1408.2041, 2014.

[4] CHEN R, SHI J, CHEN Y, et al. Powerlyra: Differentiated graph computation and partitioning on skewed graphs[J]. ACM Transactions on Parallel Computing (TOPC), 2019, 5(3): 1-39.

[5] PAGE L, BRIN S, MOTWANI R, et al. The PageRank citation ranking: Bringing order to the web.[R]. Stanford InfoLab, 1999.

[6] ZHOU A C, SHEN B, XIAO Y, et al. Cost-aware partitioning for efficient large graph processing in geo-distributed datacenters[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(7): 1707-1723.

[7] MAYER C, TARIQ M A, LI C, et al. Graph: Heterogeneity-aware graph computation with adaptive partitioning[C]//2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS). 2016: 118-128.

[8] XU N, CUI B, CHEN L, et al. Heterogeneous environment aware streaming graph partitioning[J]. IEEE Transactions on Knowledge and Data Engineering, 2014, 27(6): 1560-1572.

[9] ZHOU A C, IBRAHIM S, HE B. On achieving efficient data transfer for graph processing in geo-distributed datacenters[C]//2017 IEEE 37th international conference on distributed computing systems (ICDCS). 2017: 1397-1407.

[10] MALEWICZ G, AUSTERN M H, BIK A J, et al. Pregel: a system for large-scale graph processing[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. 2010: 135-146.

[11] TSOURAKAKIS C, GKANTSIDIS C, RADUNOVIC B, et al. Fennel: Streaming graph partitioning for massive scale graphs[C]//Proceedings of the 7th ACM international conference on Web search and data mining. 2014: 333-342.

[12] MOFRAD M H, MELHEM R, HAMMOUD M. Revolver: vertex-centric graph partitioning using reinforcement learning[C]//2018 IEEE 11th International Conference on Cloud Computing (CLOUD). 2018: 818-821.

[13] VALIANT L G. A bridging model for parallel computation[J]. Communications of the ACM, 1990, 33(8): 103-111.