分　数：	
评卷人：	

華中科技大學

研究生课程论文（报告）

课程名称：数据中心技术

题　目：基于随机游走的图处理系统

学　　号　M202273529

姓　　名　李真理

专　　业　计算机技术

课程指导教师　施展、童薇、胡燏翀

院（系、所）　武汉光电国家研究中心

2023 年 1 月 12 日

基于随机游走的图处理系统

李真理 [1)]

[1)](华中科技大学 武汉光电国家研究中心，湖北省武汉市 435400)

摘 要 图处理系统最近在学术界和工业界引起了越来越多的兴趣，因为它们在各种应用中具有显著的有效性。图随机游走是许多图处理和图学习应用程序不可或缺的主力，大多数现有的图处理算法和系统都能够处理具有数十万或者数百万节点的网络，然而如何将它们扩展到具有数千万甚至上亿的网络仍然是一个具有挑战性的问题。此外，另一个突出的问题是图随机游走系统越发难以匹配 GPU 端节点嵌入训练速度。本文调研了近几年随机游走系统针对大规模图和提升性能方面做出的优化。一些研究认为现在方法未能有效利用现代 CPU 内存层级结构，因为人们普遍认为随机游走中固有的随机性和图的倾斜性质时大多数内存访问随机，从而针对随机游走中数据局部性进行优化，如 FlashMob、ThunderRW；一些研究认为现有系统在分块加载大规模图到内存时存在大量随机 I/O 限制了随机游走的效率，从而提出新的 I/O 模型，如 GraphWalker；还有一些工作则是适普性地提升了复杂随机游走算法采样边的性能，如 KnightKing。

关键词 图；随机游走；I/O；缓存；

Graph Processing System Based on Random Walk

Li Zhen-Li[1)]

[1)](Wuhan National Laboratory for Optoelectronic, Huazhong University of Science and Technology, Wuhan, Hubei)

Abstract Graph processing systems have recently attracted increasing interest in both academia and industry due to their remarkable effectiveness in various applications. Graph random walks are an indispensable workhorse for many graph processing and graph learning applications. Most existing graph processing algorithms and systems are capable of handling networks with hundreds of thousands or millions of nodes. However, how do they scale to Networks with tens of millions or even hundreds of millions are still a challenging problem. In addition, another outstanding problem is that it is increasingly difficult for the graph random walk system to match the GPU end node embedding training speed. This paper investigates the optimization of random walk systems for large-scale graphs and performance improvement in recent years. Some research argues that current approaches fail to make efficient use of modern CPU memory hierarchies, as it is widely believed that the inherent randomness in random walks and the skewed nature of the graph make most memory accesses random, thereby optimizing for data locality in random walks , such as FlashMob, ThunderRW; some studies believe that the existing system has a large number of random I/Os when loading large-scale graphs into memory in blocks, which limits the efficiency of random walks, thus proposing new I/O models, such as GraphWalker; and Some works generally improve the performance of sampling edges of complex random walk algorithms, such as KnightKing.

Key words Graph; Random Walk; I/O; Cache;

1 引言

随机游走是一项基本且广泛使用的图形处理任务，作为从图实体之间的路径中提取信息的强大数据工具，它构成了许多重要的图测量、排序和嵌入算法的基础。例如 PageRank、SimRank、DeepWalk、node2vec，等等。这些算法可以独立工作也可以作为机器学习任务的预处理步骤。从直观上看，随机游走的大部分计算都集中在边缘采样过程中，这也体现了各个随机游走算法之间的差异。随着随机游走变得越来越流行，采样逻辑也变得越来越复杂，其中边转移概率取还与它访问过的前多个顶点相关。因此，复杂的随机游走算法以采样复杂性为代价实现了更灵活的、特定于应用程序的游走。对于 node2vec 这类高阶随机游走算法，会因为边缘采样而陷入困境，产生巨大的采样成本：在每一步，出边选择都需要重新计算所有出边的转移概率，其开销随着 walker 当前所在顶点的度而增长。

如今，许多图随机游走执行将边或路径样本提供给图嵌入训练，通常使用随机梯度下降的方法。因此，现代图嵌入框架同时在 CPU 上执行图随机游走和在 GPU 上进行嵌入训练。构建带 GPU 的嵌入系统的主要挑战包括：

（!）有限的 GPU 内存。节点嵌入的参数矩阵很大，而耽搁 GPU 的内存很小。现代 GPU 通常有 12GB 或者 16GB 的容量。

（2）有限的总线带宽。总线的带宽比 GPU 的计算速度要慢得多。如果 GPU 频繁地与主存交换数据，将会有严重的延迟。

（3）同步开销大。大量的数据在 CPU 和 GPU 之间传输。CPU 与 GPU 间或 GPU 间同步开销非常昂贵。

Zhu 等人提出了一种名为 GraphVite[1]的高性能 CPU-GPU 混合系统，用于在大规模网络上训练节点嵌入。GraphVite 通过节点嵌入算法和系统的协同优化，充分利用了 CPU 和 GPU。

GPU 相对于 CPU 的计算能力增长更快，这给主机端的随机游走带来了持续的压力，以跟上 GPU 端嵌入计的步伐。这种压力因机器学习应用普遍采用多 GPU 节点而进一步加剧，其中一些 GPU 可以使用不同的超参数同时训练独立的图嵌入。因此，今年来针对如何提高主机端的随机游走性能提出了许多效果卓越的工作。

Yang 等人提出了 KnightKing[2]降低了高阶游走的边采样的开销。这是图随机游走的第一个通用框架，它可以被看做是一个“分布式随机游走引擎”。

KnightKing 的效率和可扩展性的核心是其快速选择下一条边的能力。它的主要贡献：

（1）KnightKing 是以第一个通用的分布式图随机游走引擎；

（2）它提供了一个直观的以 walker 为中心的计算模型；

（3）提出了一个统一的边转移概率定义；（4）提出新颖的基于拒绝的采样方案，显著降低了昂贵的高阶动态随机游走算法的开销。

此外，随着互联网的兴起，现在的图网络越来越庞大，在一些互联网公司中，例如 Facebook 的用户网络、淘宝的商品网络，可能达到上亿顶点、十亿边的数量级。为了提高在单机上分析大图的性能，提出了许多核外图处理系统。这些系统的主要工作是减少随机磁盘 I/O。通常，当图太大而无法放入内存时，这些系统会将整个图划分为许多子图，并将每个子图作为一个块存储在磁盘上。为了进行图分析，以前的工作是基于迭代的模型。在每次迭代中，块被顺序加载到内存中，然后执行与加载的子图相关的分析。这样，它将大量的随机 I/O 转化为一些列顺序 I/O，并保证在每次迭代中对所有块进行同步分析。

Wang 等人的研究中观察到当前具有基于迭代模型的图系统无法有效地支持随机游走。主要限制是三个方面：

（1）由于高度随机性，许多随机游走不均匀地分布在图的不同部分，因此一些子图可能只包含很少的 walker。然而，基于迭代的模型不知道这些 walker 的状态，只是将所有需要的子图顺序加载到内存中进行分析，因此导致 I/O 利用率非常低；

（2）由于基于迭代的模型确保了同步分析，因此所有 walker 在每次迭代中只移动一步。结果，walker 更新效率也受到限制，从而进一步加剧了 I/O 效率的降低；

（3）由于游走的以随机性，每个顶点的游走次数是动态变化的，因此现的图系统通常使用海量的动态数组来记录当前经过图中每条边或每个顶点的次数。然而，这种索引设计需要很大的内存空间，从而限制了处理非常大的图的可扩展性。

为了解决 I/O 效率问题以有效快速和可扩展的随机游走，Wang 等人开发了 GraphWalker[3]，这是

一个 I/O 高效且资源友好的图系统。GraphWalker 主要贡献如下：

（1）Wang 等人开发了一种新颖的状态感知 I/O 模型，它利用每次随机游走的状态优先将具有最多游走的图块从磁盘加载到内存中，从而提高 I/O 利用率。同时，还提出了一种 walker 感知缓存方案来提高缓存效率。

（2）采用基于重入方法的异步游走更新方案，允许每次游走移动尽可能多的步数，以充分利用加载的子图，大大加快随机游走的进度。为了解决由异步更新引起的掉队问题，还采用概率方法来平衡每次行走的进度。

（3）提出了一种轻量级的以块为中心的索引方案来管理游走状态，并采用固定长度的步行缓冲策略来减少记录游走状态的内存成本。此外，还开发了一种基于磁盘的 walk 管理方案，并使用异步批处理 I/O 将行走状态写回磁盘，以支持在巨大的图上运行大量随机游走。

解决了大图的 I/O 问题，针对随机游走的性能提升，也有许多工作从不同角度做出了显著的贡献。现代研究的普遍共识是：不规则的内存访问是导致随机游走性能限制的一个普遍的问题。如何解决这个问题人们从不同方向提出了解决方案。

Sun 等人认为，对于 GC、NCP 等图分析任务，一般随机游走查询是最主要成本。因此，加速随机游走查询是一个重要的问题。由于不规则的内存访问，常见的随机游走算法有高达 73.1%的 CPU 流水线插槽停滞，这比传统的图工作负载遭受更多的内存停滞，这比传统工作负载遭受的内存停滞要多得多。因此，CPU 频繁等待对主内存的高延迟访问，这成为主要的性能瓶颈。

为了解决这个问题，Sun[4]等人设计一个通用的以 step 为中心的编程模型来抽象不同的随机游走算法。其次，基于编程模型，提出了 step 交错技术来解决由软件预取的不规则内存访问引起的缓存停顿。由于现代 CPU 可以同时处理多个内存访问请求，step 交错的核心思想是通过发出多个未完成的内存访问来隐藏访问延迟，这利用了不同随机游走查询之间的内存级并行性。

另一方面，Yang 等人从缓存的角度提出了解决方案。现代的计算机具有负载的内存层级结构，专为数据密集型应用而设计：多级 CPU 缓存、高 DRAM 带宽以及硬件预取和内存缓冲区等功能，显著地帮助程序从时间和空间局部性中获益。但是，图随机游走在大型工作集上大量执行随机访问，从最先进的 CPU 硬件中获得的收益要低得多，大部分执行时间花在数据停顿上，浪费昂贵的数据中心或服务器资源。

虽然在大图中游走具有明显的随机性，但是通过仔细的分区、重新安排和批处理操作，可以获得大量的空间和时间的局部性。Yang[5]等人提出了 FlashMob，一种新的图随机游走设计。它支持大部分顺序内存访问，以图划分进行战略性切割，以便在不同缓存级别内进行处理。

FlashMob 对包含相似度顶点的分区进行流式处理，随后实现了多重优化。对于少量的高度顶点，通过对碰巧位于这些热点上的 walker 进行批处理来获取它们的访问密度。对于低度顶点，利用它们的度数规律性，采用直接索引的简化数据结构来减少处理过程中的随机内存访问。

FlashMob 是第一个将大部分计算明确放入 CPU 缓存中的图随机游走系统，并可快速从 DRAM 流入\流出。它现在的基线相比，带来了一个数量级的性能改进。

综上，现有的工作以及它们的贡献总结如下：

（1）GraphVite 一种用于训练节点嵌入的高性能 GPU-GPU 混合系统，通过算法和系统协同优化，实现了对超大规模的图嵌入。

为了跟上 GPU 端嵌入能力的高速增长，人们提出了许多效果显著的提升 CPU 端随机游走性能的工作。

（2）KnightKing 是一个通用的分布式图随机游走引擎，以创新的基于拒绝的采样机制为中心，显著降低高阶游走算法的成本。

（3）GraphWalker 通过开发具有异步遍历更新的状态感知 I/O 模型来提升 I/O 效率，解决了大规模图随机游走中的 I/O 瓶颈问题。

针对不规则访问，这一最主要的随机游走性能限制诱因：

（4）ThunderRW 采用了 step 交错技术解决了由软件预取的不规则内存访问引擎的缓存停顿。

（5）FlashMob 通过仔细划分、重新排列和分批操作，是内存访问更加顺序和规则来提高缓存和内存带宽利用率，有效地利用了现代 CPU 内存层次结构。

2 背景

2.1 图随机游走基础知识

给定图 $G=(V,E)$ 和起始顶点 $u \in V$，随机游走 walker w 进行如下过程： u 的每个邻居 v 都与一个转移概率相关联，该转移概率决定了v 被选为下一个顶点的可能性。通常，转移概率$p(v|u)$仅取决于 u，这种情况称作一阶随机游走。在高阶随机游走中，概率以$p(v|u, t, s, \ldots)$的形式指定，其中s，t是当前游走中u的前几个节点，涉及w在计算出边的转移概率。w 根据此概率对边进行采样并重复，直到满足某些终止条件。终止可以是确定性的（在给定的步数之后）或随机的（步行者在每一步以固定概率退出）。

虽然随机游走被广泛使用，但在这里我们将详细介绍一个应用：节点嵌入，这是当今图学习的重要组成部分。给定图 $G=(V,E)$ 和维数 $d \ll |V|$，节点嵌入问题试图将每个 $v \in V$ 表示为 d 维向量 $Emb(v)$，使得 G 的结构信息被保留。通俗来说，如果节点对u和v具有“相似”的邻域，那么它们的嵌入$Emb(u)$和$Emb(v)$也彼此接近。相反，具有不同邻域的两个节点将具有更远的嵌入。这是通过在正节点对 P 和负节点对 N 的集合上训练深度学习模型来学习节点嵌入来实现的，这样 $(u, v) \in$ P 的节点嵌入彼此更接近，而 $(u, v) \in$ N 的节点嵌入彼此相距较远。通常，N 是通过随机选择节点对构建的（考虑到现实世界图的稀疏性，它们不太可能连接），而 P 是通过随机游走构建的。

广泛使用的节点嵌入算法，如 DeepWalk 和 node2vec ，其不同之处在于它们如何使用随机游走来衡量邻域相似性，通过给出不同的转移概率定义。 更具体地说，DeepWalk 执行一阶均匀随机游走。 另一方面，Node2vec 是一种具有超参数的二阶算法，用于在 BFS 和 DFS 之间创建可配置的插值。 在典型的运行中，两种算法都采用默认参数，从图中的每个节点开始，需要 10 次随机游走，长度分别为 40 和 80。

2.2 统一的随机游走算法定义

一般一般随机游走的过程为：在给定图上出发一定数量的随机游走，每个出发于一个特定的节点，每条 walker 重复在当前节点的邻居节点中随机选择一个节点，并转发到该节点。不同的随机游走算法的不同就在于邻居节点的选择。

根据当前节点的邻居节点们被选择的机会是不是相同的，可以分为无偏随机游走，即当前节点的各个邻居节点被选择的概率是相等的，和有偏随机游走，即当前节点的各个邻居节点被选择的概率取决于其权重或其他特征，各不相同。

根据各个 walker 转发过程中在一条边上的转移概率是不是恒定的，又可以分为静态随机游走，即 walker 在一条边上的转移概率只与图结构有关，在 walker 的转发过程中保持恒定，和动态随机游走，即 walker 在一条边上的转移概率不仅与图结构有关，还与 walker 当前的状态有关，会随着 walker 的转发不断变化。所以对于动态随机游走来说，我们不再能够预计算出每条边的转移概率，而在每一步 walker 转发时都需要重新计算转移概率。

我们可以进一步根据随机游走算法的转移概率考虑当 walker 最近路径的步长数来定义随机游走算法的阶数，一阶随机游走算法中，walker 只知道当前节点，不知道之前访问的节点信息。一阶随机游走算法中，walker 在选择下一跳的时候考虑上一步所在的节点。高于二阶（包含二阶）的高阶随机游走为动态随机游走。

除了转移概率的不同，不同的随机游走算法也有不同的终止策略，常见的终止策略有(1)在 walker 走到一定步长后终止，(2) 每个 walker 在每一步有一定的概率终止。

算法 1. 执行随机游走算法范式
输入：图 G 和随机游走查询集合 Q
输出：Q 中每个查询的游走序列
1. FOREACH $q \in Q$ DO
2. DO
3. 随机选择一个属于 $Q.cur$ 的邻居
4. 添加选择的点到 q
5. WHILE $Terminate(q) == false$
6. RETURN Q

2.3 随机游走基础算法

随机游走算法通常遵循算法 1 中的执行范式。它们的主要区别在于邻居选择步骤。下面介绍四种具有代表性的 RW 算法，这些算法已经在很多应用中得到应用。

*PPR（Personalized PageRank）*根据给定源v为图中的每个顶点v 分配一个分数的个性化视图，它描述了v 对v 感兴趣（或相似）的程度。 这个问

题的一个常见解决方案是从v开始一些随机游走查询，这些查询在每一步都有一个固定的终止概率，并根据随机游走查询结束顶点的分布来近似计算分数。这些算法通常将随机游走查询设置为无偏。

DeepWalk 是一种广泛应用于机器学习的图嵌入技术。 它是基于 SkipGram 模型开发的。对于每个顶点，它启动指定数量的具有目标长度的随机游走查询以生成嵌入。 初的 DeepWalk 是无偏的，而最近的工作将其扩展为考虑边缘权重，这变成了有偏（静态）随机游走。

Node2Vec 是一种流行的基于二阶随机游走的图嵌入技术。与 DeepWalk 不同的是，它的转移概率取决于最后访问的顶点。假设$Q.cur$是v。 等式 1 描述了选择边 $e\,(v, v')$ 的转移概率，其中 u 是最后访问的顶点，$dist\ (v', u)$ 是 v' 和 u 之间的距离，a 和 b 是控制随机游走行为。 Node2Vec 是动态的，因为转换概率依赖于查询的状态。 此外，它可以通过将 $p\,(e)$ 与 w_e 相乘来考虑边缘权重。

$$p(e(v,v')) = \begin{cases} \frac{1}{a} & \text{if } dist(v',u) = 0, \\ 1 & \text{if } dist(v',u) = 1, \\ \frac{1}{b} & \text{if } dist(v',u) = 2. \end{cases} \quad (1)$$

Meta-path 用于捕捉节点和边之间的异构性背后蕴含的语义。Meta-path 中，每条 walker 关联着一个 meta-path 模式，指定一个 walker 路径中边类型的模式。比如在一个论文发表网络图中，为了探索论文之间的引用关系，我们可以设置 walker 的初始节点为一个作者，设置 meta-path 模式为“isAuther -> citeBy -> autheredBy”。

2.4 处理器中的缓存层次结构

最近，出现了许多针对大量使用虚拟化的数据中心工作负载的架构创新。 虽然我们在这里讨论的是英特尔当前一代的处理器，但其他主要供应商的处理器也具有类似的特性。

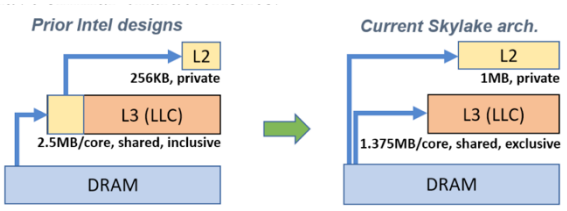


图 1 Intel 处理器缓存设计中的挑战

如图 1 所示，最近的英特尔处理器保留了现代多核处理器的分层缓存层次结构：每个内核都有其专用的 L1 和 L2 缓存，而插槽中的所有内核共享一个 LLC（末级缓存，通常是 L3） 位于内核和主内存之间。 之前的 Intel CPU（Broadwell 系列及更早版本）采用的 LLC 比 L2 大一个数量级，具有包容性 LLC 管理。这意味着所有带入小得多的 L2 缓存中的数据也驻留在 L3 中。 使用英特尔当前的可扩展系列处理器，L3 和 L2 之间的相对大小比率显着降低：例如，典型的每核缓存配置为 1MB L2 和 1.375MB L3（L3 总大小除以内核数）。新的独家 L3 设计进一步提升了显着更大的 L2：缓存未命中会将数据直接带入 L2 而不是 L3，L3 用于保存从 L2 逐出的数据，从而实现更好的整体缓存容量利用率并促进内核之间的数据共享 . 此外，通过非包容性 L3 设计，尽管组合缓存大小实际上比以前的 Broadwell 架构缩小了（每个内核从 2.5MB 到 1.375MB），但更多此类空间现在位于更快的 L2 上并保存一组不相交的数据 作为共享 L3。

虽然这些设计更改针对虚拟化和多线程工作负载进行了优化，其中更大的私有 L2 可容纳更多核心私有数据并减少工作负载间的干扰，但它们也给单一工作负载执行带来了新的机会。 通过同时执行随机访问和流式访问（以及缓存感知数据组织和工作分区），我们允许前者有意识地将其工作集保留在 L2（甚至 L1）内，而后者享受大部分共享的 L3 容量和内存带宽。

表 1　从内存层次结构中加载的延迟

位置	L1C	L2C	L3C	本地内存	远端内存
顺序读	0.42ns	0.41ns	0.44ns	0.76ns	1.51ns
随机读	0.77ns	0.95ns	2.60ns	18.35ns	24.35ns
指针追逐	1.69ns	5.26ns	19.26ns	116.90ns	194.26ns

在具有上述架构（Intel Xeon Gold 6126）的服务器上，Yang 和 Ma 等人测量了从内存层次结构的不同位置加载单个单词的延迟，具有不同的访问模式。 表 1 中的结果证实：(1) 尽管这种内存硬件具有随机访问特性，但顺序访问和随机访问之间存在很大的延迟差距； (2) 顺序流甚至从跨 NUMA 节点的远程内存带来可承受的延迟，而顺序-随机性能差距随着我们向下层次结构快速增长； (3) 指针追逐是昂贵的，其成本使得 L3 高速缓存内的访问比对 DRAM 的简单随机访问慢。

3 研究现状

3.1 图随机游走通用框架 KnightKing

KnightKing 是图随机游走的第一个通用框架。它可以被看作是一个“分布式随机游走引擎”，采用以步行者为中心的观点，使用 API 来定义自定义的边缘转换概率，同时处理常见的随机游走基本功能。与图引擎类似，KnightKing 在图分区、顶点分配、节点间通信和负载平衡方面隐藏了系统细节。因此，它促进了直观的“像 walker 一样思考”的视图，用户可以灵活地添加可选的优化。

KnightKing 的效率和可扩展性的核心是其快速选择下一条边的能力。其提出了一个统一的转移概率定义，它允许用户直观地定义自定义的随机游走算法的静态和动态转移概率。基于这个算法定义框架，KnightKing 成为了第一个执行拒绝采样的随机游走系统，消除了扫描 walker 当前所在顶点所有出边以重新计算它们的转移概率的需要。直观上，这个步骤对于动态随机游走是必要的，因为如果不评估其对于兄弟节点的转移概率，就无法继续对一条边进行采样。然而，通过拒绝采样，仅用少数几个单独快速评估的试验取代了顶点 v 处的这种完整的 $O(|E_v|)$ 检查。

3.1.1 转移概率统一定义

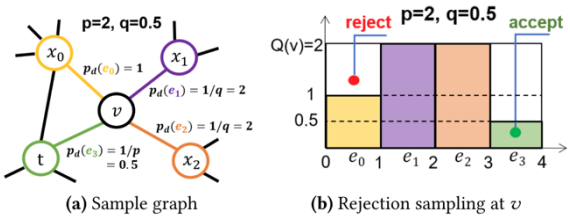
Yang 等人给出各类随机游走算法一个统一的非标准化的转移概率的定义：对于一个 walker w，当前停留在节点 v，该 walker 通过 v 的一条出边转发到下一条的概率为：

$$P(e) = P_s(e) \cdot P_d(e, v, w) \cdot P_e(v, w).$$

其中，$P_s(e)$为静态分量，$P_d(e,\ v,\ w)$为动态分量，$P_e(v, w)$为扩展分量。在这个统一定义的框架下，简单的随机游走算法就是有偏高阶算法的一个特例。比如简单的无偏静态算法，$P_s(e) = P_d(e, v, w) = 1$。$P_e(v, w)$ 的设置独立于 $P_s(e)$ 和 $P_d(e, v, w)$，当一个 walker 满足终止条件时，设置 $P_e(v, w) = 0$。

3.1.2 拒绝采样方法

此小节介绍 KnightKing 中的关键创新：其统一的边缘采样机制可以轻松处理昂贵的动态高阶游走，同时在静态游走中自动转换为别名表解决方案。以拒绝采样为中心，只需要计算几条边的非归一化转移概率。即使在百万边和顶点，它可以在不损失正确性的情况下提供快速采样。



(a) Sample graph (b) Rejection sampling at v

图 2 无偏 node2vec 的拒绝采样

考虑图 2（a）中所示的示例图上的无偏 node2vec。拒绝采样的基本思想是找到一个覆盖所有条形的包络 $Q(v)$，并将边之间的一维采样问题转换为包络下覆盖区域内的二位采样问题。在 KnightKing 中，将 $Q(v)$定义为每个顶点的常量，直观地将其绘制为与所有边上的最高值 P_d 匹配的水平线。在这种情况，$Q(v) = max(1/p,1,1/q) = 2$，例如图 2（b）。

为了对一条边进行采样，随机采样一个位置 (x, y)，该位置在由线 $y = Q(v)$ 和 $x = |E_v|$ 以及 x 和 y 轴覆盖的矩形区域内均匀分布。即，想象一下在该矩形内投掷飞镖。采样的 x 值给出候选边 e，而 y 值将与其对应的 P_d 进行比较。如果 $y \leqslant P_d$ (e)（飞镖击中木条），则认为 e 为成功样本； 否则（飞镖未击中任何木条），e 被拒绝，这需要下一次抽样试验，直到成功。

该方法的优点在于动态随机游走，其中转移概率 P_d 取决于 walker 状态并且不能有效地预先计算，这种基于拒绝的采样允许先采样，然后检查采样是否接受。这个看似微小的差异消除了跟新当前顶点所需的昂贵的所有边扫描。

3.2 高效 I/O 随机游走系统 GraphWalker

本节介绍 GraphWalker 的主要思想，它是一种针对单机随机游走的 I/O 高效和资源友好的设计。

Wang 等人的目标不仅是支持非常大量的游走，比如数百亿次的游走，而且还支持非常长的游走，比如每次游走数千步。为了实现这一目标，主要思想是采用状态感知模型，该模型利用每次游走的状态，例如，游走停留的当前顶点。总之，与基于迭代的模型盲目顺序加载图块不同，状态感知模型选择加载包含最大游走数的图块，并使每次游走尽可能多地移动步数，直到到达加载子图的边界。这样游走可以在每次 I/O 的时候尽可能地被更新。从而

低 I/O 利用率和低游走更新率都能被有效解决。

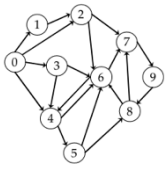


图 3 示例图

为了进一步说明上述想法并分析其好处，我们仍然考虑图 3 中的示例图。 假设我们必须运行三个从节点 0 开始的随机游走，并且必须移动四步。图 4 显示了使用状态感知模型进行图形加载和步行更新的过程。具体来说，在第一个 I/O 中，图形块 b_0 被加载到内存中，因为它包含所有三个步行。使用加载的图形块 b_0，走 w_0 和 w_1 移动两步，而 w_2 只移动一步，因为它需要其他不在内存中的图形块来走更多步。由于两次行走都落入block b_2，在第二次 I/O 中，block b_2 被加载到内存中，游走 w_0 结束，w_1 可以移动一步。最后，剩下的两次走都在块 b_1 中，所以我们将 b_1 加载到内存中，所有的走就可以完成了。值得注意的是，本例中只需要三个 I/O。 然而，对于基于迭代的模型，它可能需要 12 个 I/O，因为它使用四次迭代，并且在每次迭代中产生三个 I/O。

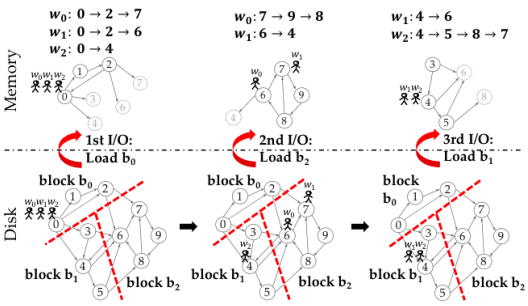


图 4 状态感知模型的主要思想

上述就是 GraphWalker 的主要设计思想，它可以支持快速且可扩展的随机游走。主要由三个部分组成（1）状态感知图加载；（2）异步游走更新；（3）以块为中心的游走管理。

3.3 高效的内存中随机游走框架 ThundrRW

Sun 等人观察到随机游走算法建立在多个随机游走查询而不是单个查询之上的。尽管查询内并行性有限，但是随机游走算法中存在丰富的查询间并行性，因为每个随机游走查询都可以独立执行。因此，ThunderRW 以 step 为中心，从查询的角度抽象了随机游走算法的计算，以利用查询间的并行性。

具体来说，从移动查询的一个 step 的本地视图对计算进行建模。然后，将一个 step 抽象为 GMU（Gather-Move-Update）操作，以表征随机游走算法的通用结构。使用以 step 为中心的模型，用户通过“think-like-a-walker”来开发随机游走算法。用户只需要专注于定义算法、设置转移概率和更新每一步状态的函数，然后 ThunderRW 将帮助用户定义的面向 step 的函数应用于随机游走查询。

3.3.1 步交错技术

ThunderRW 创新提出了步交错技术，它减少了由随机内存访问引起的流水线停顿。接下来介绍它的主要思路。

基于以步骤为中心的模型，ThunderRW 使用 GMU 操作处理查询的一个 step Q。模型下随机内存访问有两个主要来源。 首先，Move 操作随机选择一条边并沿着所选边移动Q。 其次，自定义函数中的操作会引入缓存未命中，例如 Node2Vec 中的距离检查操作。 由于用户空间中的操作（即用户定义的函数）由 RW 算法确定，并且可以非常灵活，因此我们针对系统引起的内存问题（即 Move 操作）。 受分析结果的启发，我们建议使用软件预编译技术来加速 ThunderRW 的内存计算。然而，查询步骤Q没有足够的计算工作量来隐藏内存访问延迟，因为步骤Q具有依赖关系。因此，建议通过交替执行不同查询的步骤来隐藏内存访问延迟。具体来说，给定 Move 中的一系列操作，将它们分解为多个阶段，以便一个阶段的计算使用前一阶段生成的数据，并在必要时检索后续阶段的数据。同时执行一组查询。 一旦一个查询阶段 Q 完成，就切换到组中其他查询的阶段。当其他查询阶段完成时，恢复 Q 的执行。通过这种方式，将内存访问延迟隐藏在单个查询中并让 CPU 保持忙碌。称这种方法为步交错。

图 5 给出了一个示例，其中一个步骤分为四个阶段。如果按顺序逐步执行查询，则 CPU 会因内存访问而频繁停滞。即使使用预取，阶段的计算也无法隐藏内存访问延迟。相反，步骤交错通过交替执行不同查询的步骤来隐藏内存访问延迟。

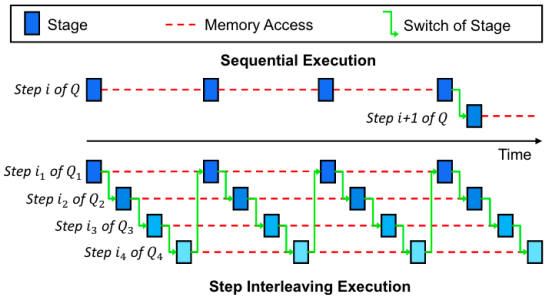


图 5 顺序交错与步交错

3.4 高效缓存随机游走系统 FlashMob

图 6 描述了 FlashMob 架构概览。重点介绍 FlashMob 架构总览：

图分区的流式处理：与现有的随机游走实现不同，FlashMob 不会跟随 walker 访问整个图，即使有足够的内存空间来承载后者。相反，它将所有顶点按度数降序排列并将它们切割成许多顶点分区。图 5 显示了两个这样的分区，一个有几个高度顶点，另一个有很多低度顶点。 一个线程一次只能处理一个任务，将当前在一个分区上的所有 walker 移动一步。 这显然需要将遍历解耦为多个阶段：在本例中为采样和混洗，向图随机遍历引入一个让人联想到 MapReduce 的流模型。

缓存友好的 walker 批处理：所有现有解决方案都单独移动 walker。FlashMob 发觉到流行顶点上的大量流量，通过其洗牌过程积极地对位于同一位置的 walker 进行批处理。 它不是为每个 walker 独立地随机获取边，而是可以对许多边进行预采样，从而实现更低开销的样本生成和充分利用缓存行，从而引入此类预采样边。

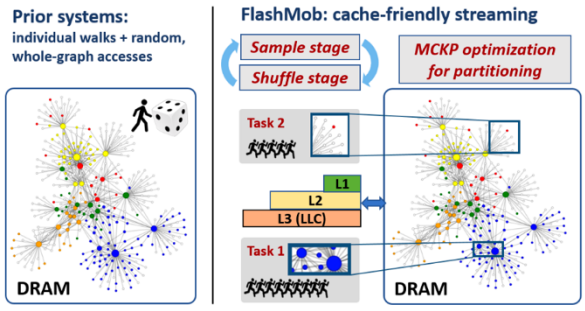


图 6 FlashMob 和 现存系统设计

FlashMob 将顶点分区与 walker 批处理相结合，通过 (1) 在缓存中拟合每个任务的工作集和 (2) 促进顺序内存访问，显着提高了内存访问效率。本质上，它在 CPU 缓存中执行“核外”处理，并使用 DRAM 进行快速数据流传输。

自动策略/分区规划：整体步行性能取决于许多因素的复杂相互作用，例如图形大小和度分布、每个缓存级别的大小/速度以及图形上的相对 walker “密度”。 FlashMob 通过近似将其优化映射到 MultiChoice KnapSack (MCKP) 问题，而不是硬编码或留给用户配置重要参数，例如分区和样本策略设置（何时启用预采样）。 如图 6 所示，它在运行开始时执行快速的一次性自动配置，将其顶点数组切割成分区，并为每个分区分配适当的样本策略。简单说明分区大小的确定过程：首先对图按照度数进行排序，分成大小相同的组，每个组内分成大小相同的分区。分区的大小只有固定几种，分别是 L1 缓存、L2 缓存、L3 缓存和 DRAM（8 倍 L3 缓存）相同大小。对于每个组内的分区，可以选择预采样和直接采样的方法，具体采样方法是通过使用预实验得到采样时间最短的采样方法来确定的。每个组的权重是这个组内的分区数，收益是针对这个组内所有分区采样时间的负值。因此，需要在分区总数不超过 P 的情况下，使得收益更大。这样，就把问题转换成了一个 MCKP 问题。

4 实验评估

由于当时缺乏通用的随机游走引擎，Yang 等人将 KnightKing 与 Gemini 的随机游走改编版本进行比较，Gemini 是用 C++实现的 2019 年最先进的分布式图计算系统。通过静态游走（DeepWalk 和 PPR），KnightKing 执行其统一的采样工作流程，但实际上没有执行拒绝采样，但是凭借系统其他方面的性能优势，KnightKing 领先 Gemini 高达 16.94 倍，这两种静态算法平均领先 8.22 倍。在动态算法方面，KnightKing 的拒绝采样具有压倒性的优势。使用传统采样的 Gemini 可以看到步行执行时间随着 Meta-path 显著增长，并随着 node2vec 爆炸增长。对于这两种算法，Twitter 图遍历都无法在 6 小时内完成。

Wang 等人将 GraphWalker 与 DrunkardMob 进行比较来验证 GraphWalker 的效率，DrunkardMob 是专门针对随机游走优化的最先进（2020 年）的单机系统。两者都是基于 GraphChi 实现的。在不同游走次数和不同数据集的所有设置下，GraphWalker 始终比 DrunkardMob 更快。在 YahooWeb 上运行 106 次游走的情况下，GraphWalker 实现了 70 倍的加速。

总的来说，GraphWalker 在所有设置下都实现了 16 到 70 倍的加速。即使在大型图上运行数百亿次随机游走，GraphWalker 仍然可以在合理的时间内完成。

Sun 等人使用 ThunderRW 实现了四种具有代表性的随机游走算法，包括 PPR、DeepWalk、Node2Vec 和 Meta-Path。结果表明，ThunderRW 的性能优于最先进的随机游走框架高达一个数量级，并且步交错将内存限制从 73.1%减少到 15.0%。

Yang 等人则是比较了 FlashMob、KnightKing 与 GraphVite。对于 DeepWalk，KnightKing 比 GraphVite 快 2.2-3.8 倍，因为它有更有效的边访问。同时，FlashMob 与 KnightKing 带来了 5.4-13.7 倍的加速，在 40ns 内完成一个游走步。对于 node2vec，FlashMob 还是比 KnightKing 实现了 3.9-19.9 倍的加速。

5 总结

GraphVite 是一种用于节点嵌入的高性能 CPU-GPU 混合系统。将现有的节点嵌入方法扩展到了 GPU，并显著加速了单台机器上的训练节点嵌入。通过并行负采样，GraphVite 可以在多个 GPU 上训练节点嵌入。

随着 GraphVite 这类多 GPU 的系统的发展，随机游走系统的性能逐渐跟不上 GPU 端的性能。因此出现了各种有效的工作从采样方法、IO、缓存等方面提升随机游走的性能。

KnightKing 是第一个通用的分布式图随机游走引擎。它提供一个统一的边转移概率定义，适用于流行的已知算法，以新颖的基于拒绝的采样方案，可显著降低昂贵的高阶随机游走算法的成本。无论当前顶点的出边数量如何，都可以在精确边采样中实现中实现接近 O(1)的复杂度，而不会损失精确性。

GraphWalker 从 IO 入手，它是一种 IO 高效系统，用于支持在单个机器上对大型图进行快速且可扩展的随机游走。GraphWalker 仔细管理图数据和游走索引，并通过使用状态感知和图加载和异步游走更新来优化 IO 效率。

ThunderRW 从内存访问入手，它是一种高效的内存中游走引擎，用户可以在其上轻松实现自定义游走算法。设计了一个以步骤为中心的模型，从 Move 查询的一个 step 的本地视图中抽象出计算。基于该模型，提出了步交错技术，通过交替执行多个查询来隐藏内存访问延迟。

FlashMob 推翻了大型图上的随机游走需要通过随机 DRAM 访问来解决。FlashMob 设法通过仔细的分区、重新排列和批处理操作，从随机游走中利用隐藏的空间和时间局部性。FlashMob 证明了即使对于具有随机性和不规则计算的程序，现代服务器的缓存也能实现高效的缓存处理。

参 考 文 献

[1] ZHU Z, XU S, QU M, 等. GraphVite: A High-Performance CPU-GPU Hybrid System for Node Embedding[C/OL]//The World Wide Web Conference on - WWW '19. 2019: 2494-2504[2023-01-07]. http://arxiv.org/abs/1903.00757.

[2] YANG K, ZHANG M, CHEN K, 等. KnightKing: a fast distributed graph random walk engine[C/OL]//Proceedings of the 27th ACM Symposium on Operating Systems Principles. Huntsville Ontario Canada: ACM, 2019: 524-537[2023-01-07]. https://dl.acm.org/doi/10.1145/3341301.3359634.

[3] WANG R, LI Y, XIE H, 等. GraphWalker: An I/O-Efficient and Resource-Friendly Graph Analytic System for Fast and Scalable Random Walks[J].

[4] SUN S, CHEN Y, LU S, 等. ThunderRW: An In-Memory Graph Random Walk Engine[J].

[5] YANG K, MA X, THIRUMURUGANATHAN S, 等. Random Walks on Huge Graphs at Cache Efficiency[C/OL]//Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles CD-ROM. Virtual Event Germany: ACM, 2021: 311-326[2023-01-07]. https://dl.acm.org/doi/10.1145/3477132.3483575.