计算机科学
COMPUTER SCIENCE

面向幂律图的动态图存储结构Power-PCSR

毛志雄, 刘志楠, 高叙宁, 王蒙湘, 巩树凤, 张岩峰

引用本文

毛志雄, 刘志楠, 高叙宁, 王蒙湘, 巩树凤, 张岩峰. 面向幂律图的动态图存储结构Power-PCSR[J]. 计算机科学, 2024, 51(8): 56-62.

MAO Zhixiong, LIU Zhinan, GAO Xuning, WANG Mengxiang, GONG Shufeng, ZHANG Yanfeng. Power-PCSR:An Efficient Dynamic Graph Storage Structure for Power-law Graphs [J]. Computer Science, 2024, 51(8): 56-62.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231000155

面向幂律图的动态图存储结构 Power-PCSR

毛志雄[1] 刘志楠[1] 高叙宁[2] 王蒙湘[3] 巩树凤[1] 张岩峰[1]
1 东北大学计算机科学与工程学院 沈阳 110167
2 东北大学医学与生物信息工程学院 沈阳 110167
3 中国标准化研究院 北京 100088
(20215949@stu.neu.edu.cn)

摘 要 图数据在现实生活中广泛存在,且不断发生变化。传统高效的静态图存储方式——压缩行/列(Compressed Sparse Row/Column,CSR/CSR)存储方式在更新图数据时需要大量的数据迁移,不适用于动态图数据。而能够高效更新图数据的邻接表(Adjacency List,AL)存储方式往往带有大量的指针,导致其图数据读取和分析效率低。Packed Compressed Sparse Row(PCSR)是一种基于CSR的动态图存储结构。该结构在存储边数据时并不是采用连续无空隙数组,而是采用留有空槽的压缩存储阵列(Packed Memory Arrays,PMA)结构,便于边数据的插入。因此,PCSR支持高效图更新和图分析。但是,PCSR在存储幂律图时,其性能容易受大度数顶点的影响。为此,基于PCSR提出一种支持可高效更新和分析动态幂律图的图存储结构Power-PCSR。该结构将幂律图中度数较大的顶点单独存储在一个独立的PMA中,其他所有小度数顶点与PCSR一样存储在原PMA中。小度顶点变化导致的数据迁移不会触及大度数顶点,从而大大减少了数据迁移数量;同样,大度数顶点更新导致的数据迁移只限制在每个大度数顶点的PMA内部,不会涉及小度数顶点和其他大度数顶点的数据迁移。实验显示,Power-PCSR在分析图数据时与PCSR具有相似的性能,而在更新图数据时比PCSR快2倍。

关键词:动态图存储;动态图更新;数据迁移;Power-PCSR;幂律图
中图分类号 TP391

Power-PCSR:An Efficient Dynamic Graph Storage Structure for Power-law Graphs

MAO Zhixiong[1],LIU Zhinan[1],GAO Xuning[2],WANG Mengxiang[3],GONG Shufeng[1] and ZHANG Yanfeng[1]
1 School of Computer Science and Engineering,Northeastern University,Shenyang 110167,China
2 College of Medicine and Biological Information Engineering,Northeastern University,Shenyang 110167,China
3 China National Institute of Standardization,Beijing 100088,China

Abstract Graph data is widespread in real life and changes over time. The traditional efficient static graph storage structure, compressed sparse row/column(CSR/CSC) requires a large amount of data migration when inserting/deleting edges to/from graphs,which is not suitable for dynamic graphs. Although the adjacency list(AL) is able to update graphs efficiently,it is inefficient in reading and analyzing graphs since it has a large number of pointers,which results in random memory access. PCSR is a novel dynamic graph storage structure based on CSR. It employs a packed memory arrays(PMA) to store the edges rather than a continuous array. Because there are empty slots in PMA,it is easier to insert/delete edges. Thus,packed compressed sparse row (PCSR) is efficient in both graph updating and analysis. However,we find that the performance of PCSR suffers from large degree vertices when storing power-law graphs. For this,this paper proposes a new graph storage structure based on PCSR,Power-PCSR,which supports efficient updating and analysis of dynamic power-law graphs. In Power-PCSR,each large-degree vertex is stored in an independent PMA separately,and other vertices with small degrees are stored in a PMA. The data migration caused by the small-degree vertices will not lead to the migration of large-degree vertices,thus greatly reducing the amount of data migration. Similarly,the data migration caused by the update of large-degree vertices is only limited to its PMA,and will not involve the data migration of other large-degree vertices and small-degree vertices. Experiments show that Power-PCSR has similar performance to PCSR when analyzing graphs,and is 2 times faster than PCSR when updating graph data.

Keywords Dynamic graph storage,Dynamic graph updates,Data migration,Power-PCSR,Power-law graph

到稿日期:2023-10-23 返修日期:2024-03-31
基金项目:国家自然科学基金青年科学基金(62202088)
This work was supported by the Young Scientists Fund of the National Natural Science Foundation of China(62202088).
通信作者:王蒙湘(wangmx@cnis.ac.cn)

1　引言

图结构是计算机领域中常用的一类复杂的数据结构，广泛存在于现实生活中[1]。传统的图数据存储格式通常采用压缩稀疏矩阵的方式，比如 CSR(Compressed Sparse Row)和 CSC(Compressed Sparse Column)等[2]。这是因为现实生活中的图数据往往存在一定的稀疏性，使用压缩稀疏矩阵的方式能够大量减少内存开销。此外，压缩稀疏矩阵结构在内存中顺序存储，有利于图数据的高效遍历，便于图数据的分析和处理。

然而，现实生活中的图数据在不断发生变化。比如 Twitter 在 2019 年每秒发送约 6 000 条推文[3]。中国的政务微博账户总数达到 3.7 万个。在疫情期间 3 000 多家媒体微博发布相关权威信息 607.6 万条[4]。图的高度动态性对图存储技术提出了新的挑战。在这种情况下，传统压缩矩阵的存储格式如 CSC/CSR 由于其顶点和边在内存中连续且紧密，因此在图数据进行插入或删除操作时需要移动大量的数据才能完成图更新操作。

为了能够实现图数据的高效更新，可以采用相对灵活的邻接链表等基于指针的图存储结构。邻接表(Adjacency List，AL)[5]可以通过插入或者删除指针的方式完成，避免大量数据的迁移。但是在 AL 中，由于引入大量的指针，其图数据的遍历效率严重受损，无法达到动态图数据实时更新分析结果的要求。用 AL 和 CSR 存储 Wiki-Vote 图数据数据集(7 115 个顶点，103 689 条边)时，执行图更新和图分析算法的运行时间如图 1 所示。可以看出，AL 在进行图分析时需要更多的时间，而在更新图时则非常高效。与 AL 相反，在执行图分析任务时，CSR 需要更少的时间，而在图更新时则效率较低。

为了解决传统图数据存储方式对动态图数据存储和处理性能不足的问题，许多新型的图存储结构相继被提出。Wheatman 等基于传统的 CSR 存储结构，提出了 Packed Compressed Sparse Row(PCSR)[6]。PCSR 存储结构使用了 Packed Memory Arrays(PMA)[7-8]来维护边数据。PMA 是一个带有空闲位置的连续存储空间，其空闲位置分散在存储空间中。当有新的数据插入时，其可以快速实现插入操作，避免因插入新数据而出现大量数据迁移。当执行图分析算法，对图数据进行读取时，由于图数据仍然是连续存储在内存空间中，虽然空闲位置会带来一定的冗余读取，但是其读取图数据的速度仍然非常快。

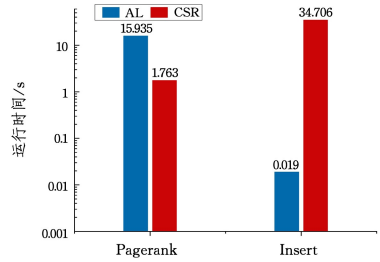


图 1　采用 CSR 与 AL 存储图数据时插入数据和执行 PageRank 所用时间

Fig. 1　Runtime of random edge insertions and executing PageRank when using CSR and AL as the storage structure

虽然 PCSR 使用了 PMA 维护图数据，一定程度上缓解了 CSR 存储图数据所导致的大量数据迁移问题，且可以较高效地处理图数据，但是图数据的频繁更新操作仍然会导致大量数据迁移(某一连续空间没有了空闲位置)，甚至会造成 PSCR 扩容(原有空间不够)。更重要的是，现实生活中的图数据往往具有幂律分布的特性[9]，即大多数的顶点只有少量的边，而极少数的顶点拥有大量的边数据，如社交网络图中关注明星的人往往异常多(社交网络中的关注视为两用户之间存在边)。幂律图中度数大的顶点往往不稳定，其变化幅度会较大，例如某一用户突然得到大量的关注或取消关注组。PCSR 空槽均衡机制处理该情况时，局部密集的插入会导致大量数据迁移甚至扩容。此外，由于数据的插入可能会导致 PMA 中空闲位置分布不均衡，大度点的存在会使存储位置相邻的顶点在空闲位置调整时导致大量的数据迁移。在处理具有局部密集性质的图数据时，PCSR 的图更新效率往往会受到一定影响。因此，将大度数顶点的邻居单独存放，分别管理，是解决 PCSR 扩容和数据迁移的一种有效方法，故采用分层数据结构。在分层设计中，每个顶点的邻居的存储位置取决于该顶点的度数。一些分层的动态图存储结构或者系统，比如 Terrace[10]，Sortledton[11]，RisGraph[12]等，将邻居少的小度数顶点用 PMA 存储，邻居多的大度数顶点用树或跳表[13-14]存储，便于大度顶点的更新。例如，Terrace 结构使用了 B 树作为索引[15]。但是，树或跳表的引入，导致大度顶点的数据遍历开销增大，从而降低图分析性能。

为了延续 PCSR 高效图数据更新和图分析的优势，本文提出 Power-PCSR——一种面向幂律图的动态图存储结构，其结构如图 2 所示。Power-PCSR 采用分层的方式，将每个大度点单独存储在一个独立的 PMA 中以提高数据更新的局部性。首先，存储小度顶点的 PMA 在进行空闲位置均衡调整时，不会因为大度点的存在而出现大量数据迁移。其次，相比树或跳表等结构没有指针的引入，继续使用 PMA 存储大度点，良好地继承了 PMA 存储的优势，具有良好的图更新和分析性能。另外，大度点将数据移动等操作限制在其独立的 PMA 内，相比 PCSR 避免了频繁地调整 PMA 内元素以及对整个 PMA 扩容的问题。

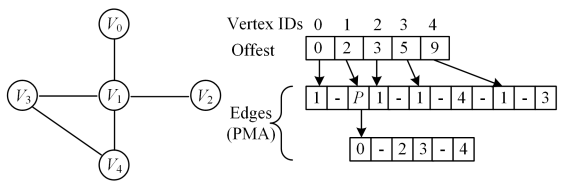


图 2　Power-PCSR 示例

Fig. 2　Example of Power-PCSR

为了支持对动态图数据的高效更新，本文还提出了延时均衡(Lazy Balance)。当有新的数据插入 PMA 中时，空闲位置会变得不均衡。为了拥有良好的插入环境，须对空闲位置进行均衡调整，使其分布均匀。空间位置的均衡操作需要对 PMA 中的数据进行迁移，其时间开销较大。延时均衡策略是在 PMA 的某个片段内因其没有任何空闲位置而无法插入新数据时才执行空闲位置均衡操作，并不是每次插入新的数据

就立即执行空闲位置均衡。

本文主要贡献如下：

(1)采用分层的结构，提出了面向幂律图的动态图数据存储结构 Power-PCSR，将每个大度顶点的边数据单独存储在一个独立的 PMA 中。延续了 PCSR 高效图数据更新和处理的优势。

(2)提出了延时均衡，减少了空闲位置的均衡操作，提升了 Power-PCSR 结构的吞吐率。

(3)实现了面向幂律图的动态图数据存储结构 Power-PCSR，并在真实数据集上与现有动态图存储结构做了性能对比。

本文第 2 章对当前的图存储结构进行简要介绍和总结；第 3 章介绍了本文所提出的动态幂律图存储结构 Power-PCSR；第 4 章通过实验证明了结构的高效性；最后总结全文。

2 图存储结构

本章简要介绍现有图存储结构。图 3 展示了图 G 及其变化后的图 G'，G' 为 G 中顶点 v_0 与 v_2 之间新增一条边。

图 3 图 G 及其变化后的图 G'

Fig. 3 Graph G and its updated version G'

2.1 邻接链表

AL 是图的链式存储结构，如图 3 所示。对图中每个顶点 v_i 建立一个单链表，v_i 的邻居存储在单链表中。图 4 为邻接链表的存储格式以及插入一条边后的结构变化示意图。从图中可以看出，添加一个邻居只需要在链表尾部插入一个节点即可。

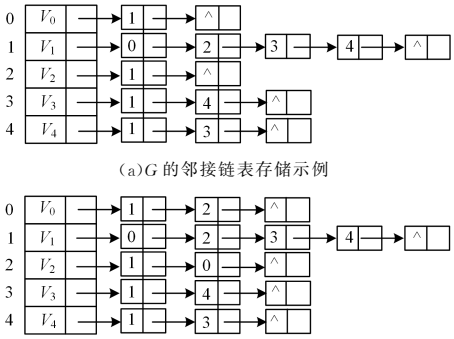


图 4 邻接链表执行图更新示例

Fig. 4 Example of graph update with AL

2.2 行/列压缩(CSR/CSC)

CSR/CSC 使用偏移数组和边数组两个数组存储稀疏图，其中 CSR 存储出邻居，而 CSC 存储入邻居。偏移数组中存储相应顶点邻居在边数组中的起始位置。图 5 展示了 CSR 的存储结构及其插入一条边后的变化示意图，从图中可以看出添加边后需要移动数据。

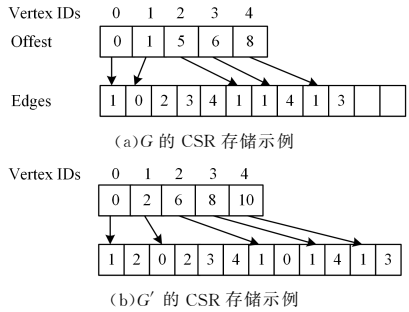


图 5 CSR 执行图更新示例

Fig. 5 Example of graph update with CSR

2.3 Packed CSR(PCSR)

PCSR 偏移数组存储顶点邻居的偏移位置(Offset)和邻居的数量(Number)，使用 Packed Memory Array(PMA)存储边数据。PMA 中的元素按照排序顺序存储，并与空槽交错。PMA 通过控制元素密度避免在每次插入或删除之后更改整个数据结构。如果某段数据变得过于密集或过于稀疏，将调整整个数据结构的大小。图 6 为 PCSR 的存储格式以及插入一条边后的结构变化示意图。从图中可以看出，添加一条边后会移动部分数据以维护结构。

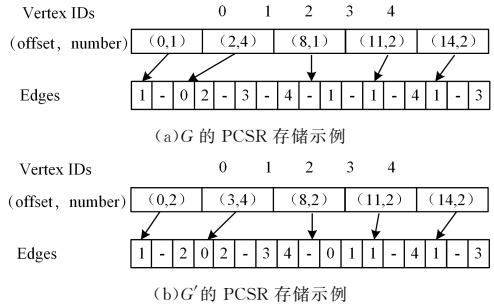


图 6 PCSR 执行图更新示例

Fig. 6 Example of graph update with PCSR

3 Power-PCSR

本章介绍所提动态图存储结构 Power-PCSR，其结构示例如图 7 所示。偏移数组与 PCSR 相同，存储顶点邻居的偏移位置和邻居的数量，边数组中 S 是哨兵，用于更新每个顶点边数据范围的起始和结束指针。与 PCSR 不同的是，边数组中邻居较少的小度顶点与邻居多的大度顶点分开存储，并不是存储在同一个 PMA 中。所有的小度顶点的邻居存储在一个 PMA 中，每个大度点的邻居单独存储在一个独立的 PMA 中，并构造索引数组，其中索引数组存储的(min, max)指索引范围内边数据的目的顶点 ID 的最小值和最大值。

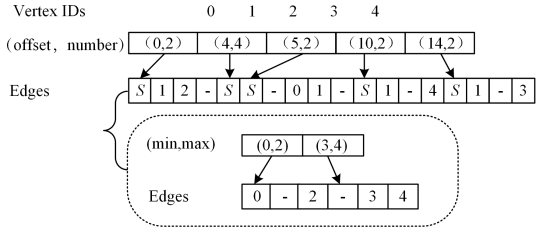


图 7 Power-PCSR 存储结构示例

Fig. 7 Example of Power-PCSR storage structure

3.1 大度点与小度点的划分

Power-PCSR 依据节点度数将大小度点采用不同的数据结构管理，但是不同的图数据类型有不同的划分界限，本文采用一种可交互的划分方法选取一定的大度点比例。统计图中不同度数的顶点的个数，然后绘制频率直方图，选取其斜率下降缓慢处为大度点与小度点的分割处。尽管图数据的规模会发生变化，但图数据中大度顶点的比例往往呈现一种固定的趋势。图 8 为 Live-Journal 数据集的顶点度数频率直方图。从图中可以看出，在顶点度数小于 75 时，顶点的数量下降平缓，即绝大部分的顶点的度数在 75 以下，少数顶点的度数在 75 以上。

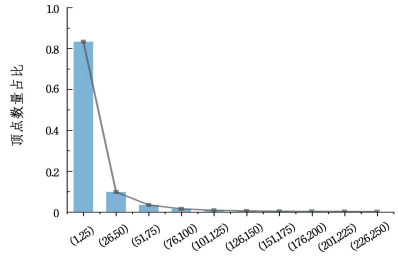


图 8 Live-Journal 顶点度数分布频率直方图

Fig. 8 Frequency histogram of Live-Journal in vertex degree

3.2 大度点与小度点的转换

在图数据变化的过程中，某些小度顶点会随着时间的推移慢慢变成大度顶点，而某些大度点也会变成小度点。当大度点变成小度点后，将其边数据逐个插入存放小度顶点的 PMA 中。由于存放小度顶点的 PMA 中仍然存有该顶点的哨兵，因此其边数据直接插入该顶点哨兵后的空槽内即可。当空槽不够时，执行空槽均衡操作，使其有空槽插入边数据。

PMA 均衡空槽以维持每个数据段元素密度时，是以 2 的倍数递增的数据段作为调整区域，故 PMA 中必含有 2 的幂次方个数据段[16]。当 PMA 存储节点数为 $|V|$ 的图数据时，开辟长度为 $2^{\lfloor lb|V|\rfloor+1}$ 的存储空间，其中 $lb(x)$ 指以 2 为底的对数操作。然后逐个插入边，同时伴随着结构的调整与扩容，直至所有边全部插入。沿用该策略处理小度点向大度点转化时，PMA 的构造问题会带来额外的结构均衡、数据插入开销，故本文在邻居数目为 $|E|$ 的小度点变成大度点时，首先构造一个独立的 PMA。PMA 的初始长度为 $2^{\lceil lb|E|\rceil}$，该长度能够在不扩容的情况下容纳现有数据且满足 PMA 结构限制，其中 $\lceil lb|E|\rceil$ 是每个数据段的长度。然后将该点的边数据拷贝到新开辟的 PMA 中而非逐个插入，尽可能地降低迁移数据带来的额外花销。独立 PMA 的索引数组存储每个 $\lceil lb|E|\rceil$ 长度的数据段的起始指针和邻居个数，如图 7 所示。每当把数据迁移到独立 PMA 时，原 PMA 中会出现大量的空槽。而这些空闲位置的连续放置不利于其他顶点边数据的插入，因此需要对其进行空槽均衡操作。处理该原因造成的空白连续数据时，本文直接移动相邻数据段的哨兵 S，从而将该段空间分配给相邻段。

3.3 延时均衡

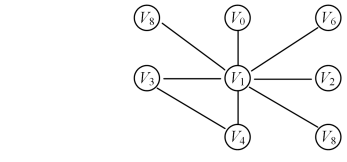
大度顶点与小度顶点的转换过程，以及数据插入与删除过程，均会导致 PMA 空槽位置分布不均衡，因此需要对空槽位置执行均衡操作。然而，空槽均衡操作需要对 PMA 中的数据进行迁移，频繁的空槽均衡操作会导致 PMA 中的数据频繁迁移，从而导致 Power-PCSR 的数据吞吐率下降。为了提升 Power-PCSR 的数据吞吐性能，本文提出了延时均衡来避免频繁空槽均衡操作所导致的大面积频繁数据迁移。

PCSR 和 PMA 要求在插入数据后对 PMA 中的空槽进行均衡调整，便于后期数据插入。频繁插入数据，容易导致频繁的空槽均衡操作。因此，在 Power-PCSR 中取消了这种插入数据后的空槽均衡调整操作，而是将该操作延迟到下次数据插入时，即当数据无法插入时才执行空槽均衡操作。延时均衡策略将多次均衡操作合并为一次操作，降低了延时均衡操作的开销。此外，延时均衡只是将空槽均衡操作的执行往后延迟，并未带来其他开销，因此延时均衡策略相比即时均衡策略会有更大的数据吞吐率。

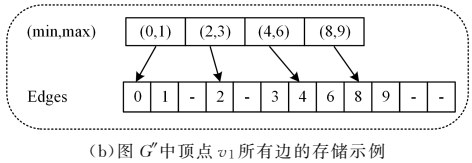
3.4 大度顶点索引

存放大度顶点的 PMA 存储了该大度顶点的所有边数据。在每次对大度顶点添加新边时，会在管理该大度顶点的整个 PMA 内进行二分查找来寻找合适的插入位置。由于 PMA 存在一定的空槽，在进行二分查找时，会出现中间位置指向空槽而无法比较的情况。此时需要先向右寻找到第一个不为空的位置，若不存在再向左寻找，将找到的位置设置为新的中间位置，再根据新的中间位置继续二分。当二分查找的规模巨大时，在中间位置左右频繁地查找第一个非空元素会带来较大的时间开销。为了减少 PMA 中的空槽对二分查找的干扰，本文提出大度点索引策略以减少寻找二分查找中间位置的花销，提升插入新边时的效率。

本文设计了一个如图 9(b)所示的边数组索引，索引数组中的每个元素表示该元素要管理的边数据的目的顶点的范围。索引数组中存有指向边数组中该范围起始存储位置和结束存储位置的指针。当对大度点添加新边时，需要先在索引数组中找到目的顶点的范围，然后在该范围指针指向的边数组范围内进行二分查找，寻找到要插入的位置。由于索引数组中数据连续存储，因此索引提供的查找不被空槽干扰，当查找一条边，索引数组和 PMA 的组合结构与单个 PMA 查询拥有相同的二分查找次数时，前者速度更快。随着边数据规模的增大，当触发大度点 PMA 的扩容时，边数组和索引数组同时扩展为原来的两倍，并且重新分配索引的目的顶点范围，更新相应的指针，从而控制每段索引所管理的边数据的数目，有利于之后的二分查找。图 9 展示了在图 G 中新增 3 条边后形成的图 G'' 以及 G'' 中 v_1 顶点所有边的存储示例，这种方式可以更加高效地访问和更新大度顶点的边数据。



(a)图 G''



(b)图 G'' 中顶点 v_1 所有边的存储示例

图 9 可拓展索引扩容示例

Fig. 9 Example of scalable index

4 实验分析

4.1 实验环境

Power-PCSR 采用 C++实现。由于原 PCSR 的开源代码中未提供边数据的删除操作，因此为测试 PCSR 和 Power-PCSR 的性能，本文实现了两者的边数据删除操作。实验设备操作系统为 Ubuntu 22.04.2 LTS，处理器为 AMD Ryzen 7 4800H，内存 16 GB。Power-PCSR 是基于 PCSR 扩展优化而来的动态图数据结构，适用于动态幂律图的图更新和分析。为了说明 Power-PCSR 的高效性，将比较 Power-PCSR 和 PCSR 在图更新和图分析上的性能。

实验数据：本文使用 4 个真实幂律图进行测试，具体信息如表 1 所列。针对这些静态图，实验过程中通过往数据集中插入和删除边数据来模拟动态图数据。

表 1 数据集

Table 1 Datasets

数据集	节点数	边数
Pokec	1 432 693	30 622 564
LiveJournal1	4 308 452	68 993 773
Hollywood	1 139 905	113 891 327
Orkut	2 997 166	106 349 209

4.2 图更新效率评估

为测试 Power-PCSR 的有效性，本节分别以随机插入、删除边数据和对大度点中的边数据执行插入、删除两种不同的图更新方式来评估 Power-PCSR 的图更新效率。

4.2.1 随机插入

首先对表 1 中的 4 个数据随机选取各自千分之一的边作为图变化，对其分别进行插入和删除操作。图 10 展示了 Power-PCSR 和 PCSR 随机插入数据时的运行时间。4 个数据集上运行的结果都表明：Power-PCSR 优于 PCSR。在随机插入时，图更新的花销主要由插入操作和空槽均衡操作造成。图 11 展示了随机插入时 PCSR 和 Power-PCSR 的数据迁移量（即在整个数据插入过程中，空槽均衡操作导致的数据迁移的次数）对比。

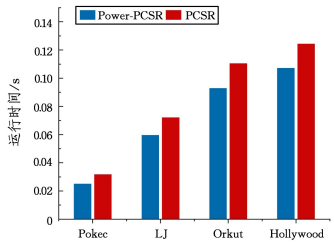


图 10 随机插入数据时运行时间的对比

Fig. 10 Comparison of running time when randomly inserting data

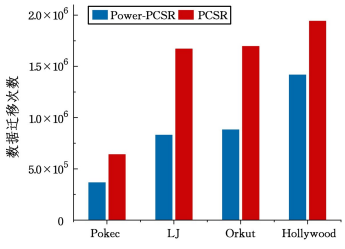


图 11 随机插入数据时迁移次数的对比

Fig. 11 Comparison of migration times when randomly inserting data

可以看出，PCSR 的数据迁移量是 Power-PCSR 的 2～3 倍。这主要是因为，PCSR 中所有的点都存储在同一个 PMA 中，空槽均衡操作极易产生大量的数据迁移操作，因此 Power-PCSR 在随机插入数据时具有更好的性能。

4.2.2 随机删除

随机删除数据时，Power-PCSR 和 PCSR 的用时如图 12 所示。可以看出，随机删除边时，Power-PCSR 与 PCSR 的时间效率几乎一样。这是因为删除边时，需要先遍历找到该条边所在位置，然后将该位置置为空。这两种数据结构置空的时间复杂度都为 O(1)，Power-PCSR 和 PCSR 的遍历效率几乎一样。这两种数据结构删除边时均不用执行空槽均衡操作，因此两种存储结构在数据删除时具有相同的性能。

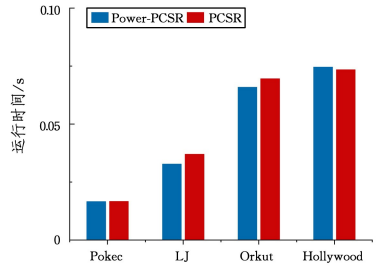


图 12 随机删除数据时运行时间的对比

Fig. 12 Comparison of running time when randomly deleting data

4.2.3 大度点插入

由于大度顶点在现实生活中通常代表一些影响力比较大的对象，因此其在真实图数据中更容易发生变化。为此，本节将测试 Power-PCSR 和 PCSR 对大度点的边执行插入和删除操作的性能。由于 Power-PCSR 和 PCSR 在删除时具有相同的性能，因此对于大度顶点的操作，本节只测试对其插入边数据时 Power-PCSR 和 PCSR 的性能。

同样，本实验仍然是对图中的大度点随机插入数据集总边数的千分之一条边进行测试。图 13 展示了大度点插入数据时运行时间的对比结果。

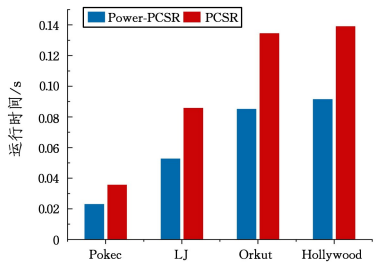


图 13 大度点插入数据时运行时间的对比

Fig. 13 Comparison of running time when inserting data into large degree vertices

可以看出，Power-PCSR 与 PCSR 在该条件下较随机插入具有明显的效率提升。Power-PCSR 在独立管理大度点的 PMA 中进行数据插入时，独立 PMA 的限制作用相比 PCSR 更不易引起整个 PMA 的其他元素进行调整和移动，Power-PCSR 极大地降低了空槽均衡开销。图 14 展示了对大度点插入数据时空槽均衡操作导致的数据迁移次数对比，可以看出，PCSR 的数据迁移量是 Power-PCSR 的 3～4 倍。此外，在大度点插入的边的数目极大时，Power-PCSR 只会对管理该大度点的 PMA 进行扩容，扩容开销远远小于 PCSR（PCSR

会对整个 PMA 进行扩容）。实验结果证明了大度节点对数据结构性能的影响，以及分层数据结构的必要性。

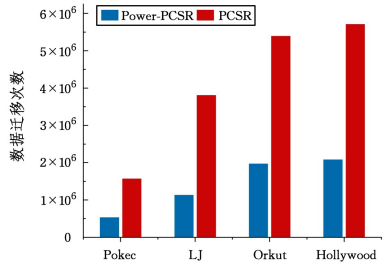


图 14　对大度点插入数据时迁移量的对比

Fig. 14　Comparison of migration amount when inserting data into large degree vertices

4.3　延时均衡有效性

为测试本文所提出的延时均衡的有效性，本实验在执行随机插入的情况下采用 Power-PCSR 存储动态图数据，分别使用延时均衡和即时均衡两种不同策略测试其图数据的更新时间。实验结果如图 15 所示，相较于即时均衡策略，延时均衡可以降低大约 1/4～1/5 的图更新时间，这是因为延时均衡策略可以将多个空槽均衡操作合并到一起执行，从而减少了空槽均衡的次数。

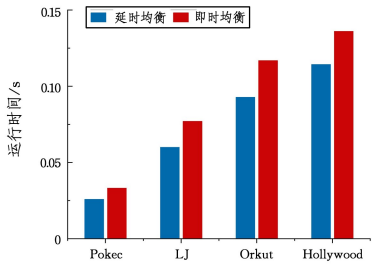


图 15　延时均衡的有效性

Fig. 15　Effectiveness of delay balance

4.4　图分析效率评估

虽然 Power-PCSR 相比 PCSR 在更新图数据时具有显著的性能提升，但是如果降低了图分析性能，那么 Power-PCSR 的应用则会受限。为此，本实验测试 Power-PCSR 在执行图分析时的性能。

本节以随机选取了 PageRank[17] 和广度优先搜索（BFS）这两种具有代表性的图分析任务来评估 Power-PCSR 的图分析效率。PageRank 算法在一定条件下，极限情况访问每个节点的概率收敛到平稳分布，这时各个节点的平稳概率值就是其 PageRank 值，该值表示节点的重要度。BFS 是找到从起点开始的最短合法路径的算法。

图 16 和图 17 分别展示了在不同数据集上以 Power-PCSR 和 PCSR 作为存储结构执行 PageRank 和 BFS 的时间。从图中可以看出，以 Power-PCSR 和 PCSR 作为存储结构时，各个数据集上执行 PageRank 和 BFS 的时间大致相同，即分层嵌套的策略对原本的遍历操作几乎无影响。由于使用了独立的 PMA，Power-PCSR 引入了指针存储管理大度点的 PMA，这部分的指针影响了执行 Page-Rank 算法的时间，使得它的遍历略慢于 PCSR。

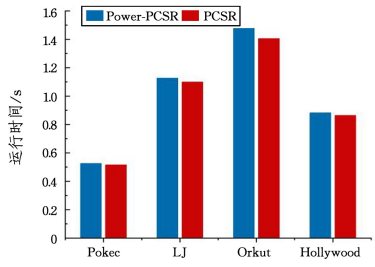


图 16　PageRank 算法的执行时间

Fig. 16　Runtime of PageRank

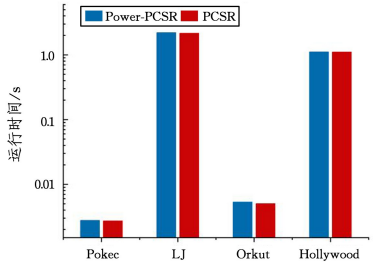


图 17　BFS 的执行时间

Fig. 17　Runtime of BFS

结束语　本文根据幂律图的特征，通过分析大度数顶点会导致 PCSR 的频繁扩容和数据迁移等问题，提出了基于分层的动态图存储结构 Power-PCSR。该结构根据节点的度数将边存储在不同的 PMA 中，即将所有小度点存储在一个 PMA 中，每个大度点单独存储在一个独立的 PMA 中。本文提出的将小度点与大度点以及大度点之间单独分离存储的方式，可以缓解在空槽均衡操作时大度节点中的大量邻居迁移导致更新慢的问题。此外，本文提出的 Power-PCSR 以较低的开销实现了大度点和小度点的动态转化；为提升 Power-PCSR 的吞吐率，提出了延时均衡策略和大度点索引，以进一步减少维护结构造成的开销。实验结果表明，Power-PCSR 在处理幂律图时具有更优异的性能。然而，在基于分层的动态图存储结构 Power-PCSR 中，随着大度点的邻居数量的增多，存储大度点的 PMA 规模会增大，进而导致图更新与图分析的性能下降。因此，下一步将研究使用树形结构以递归方式存储大度顶点的邻居，从而缓解模型性能下降问题。

参 考 文 献

[1]　YU G，GU Y，BAO Y B. Large Scale Graph Data Processing on Cloud Computing Environment[J]. Chinese Journal of Computers，2011，34(10)：1753-1767.

[2]　LANE P A，BOOTH J D. Heterogeneous sparse matrix-vector multiplication via compressed sparse row format[J]. Parallel Computing，2023，115：102997.

[3]　Episode 3 Of 4：How To Increase Engagement on Twitter[EB/OL]. https://www.sirensearch.co.uk/2019/05/23/episode-3-of-4-how-to-increase-engagement-on-twitter/.

[4]　Weibo 2020 User Development Report [R/OL]. https://data.weibo.com/report/reportDetail?id=456.

[5]　TIAN J，PANG Y. Adjoin：A causal consistency model based on the adjacency list in a distributed system[J]. Concurrency and Computation：Practice and Experience，2020，32(22)：e5835.

[6] WHEATMAN B, XU H. Packed Compressed Sparse Row: A Dynamic Graph Representation[C]// 2018 IEEE High Performance Extreme Computing Conference (HPEC). Waltham, MA, USA, 2018: 1-7.

[7] BENDER M A, EBRAHIMI R, HU H, et al. B-Trees and Cache-Oblivious B-Trees with Different-Sized Atomic Keys[J]. ACM Transactions on Database Systems, 2016, 41(3): 1-33.

[8] DE LEO D, BONCZ P. Packed Memory Arrays-Rewired[C]// 2019 IEEE 35th International Conference on Data Engineering (ICDE). Macao, China, 2019: 830-841.

[9] ZHANG S, JIANG Z, HOU X, et al. DRONE: An Efficient Distributed Subgraph-Centric Framework for Processing Large-Scale Power-law Graphs[J]. IEEE Transactions on Parallel and Distributed Systems, 2023, 34(2): 463-474.

[10] PANDEY P, WHEATMAN B, XU H, et al. Terrace: A Hierarchical Graph Container for Skewed Dynamic Graphs[C]// Proceedings of the 2021 International Conference on Management of Data (SIGMOD'21). Association for Computing Machinery, New York, NY, USA, 2021: 1372-1385.

[11] FUCHS P, MARGAN D, GICEVA J. Sortledton: a universal, transactional graph data structure[J]. The VLDB Endowment, 2022, 15(6): 1173-1186.

[12] FENG G, MA Z, LI D, et al. RisGraph: A Real-Time Streaming System for Evolving Graphs to Support Sub-millisecond Per-update Analysis at Millions Ops/s[C]// Proceedings of the 2021 International Conference on Management of Data (SIGMOD'21). Association for Computing Machinery, 2021: 513-527.

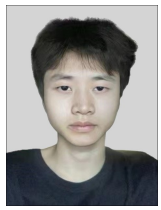
[13] SAFAEE S, MIRABI M, RAHMANI A M, et al. A distributed B+ Tree indexing method for processing range queries over streaming data[J]. Cluster Computing, 2024, 27(3): 1251-1274.

[14] HE J, YAO S, CAI L, et al. SLC-index: A scalable skip list-based index for cloud data processing[J]. Journal of Central South University, 2018, 25(10): 2438-2450.

[15] CHANG J W, CHEN T Y. When B-Tree Meets Skyrmion Memory: How Skyrmion Memory Affects an Indexing Scheme[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 41(11): 3814-3825.

[16] BENDER M A, DEMAINE E D, et al. Cache-oblivious B-trees[J]. SIAM Journal on Computing, 2005, 35(2): 341-358.

[17] ARRIGO F, HIGHAM D J, NOFERINI V. Non-backtracking pagerank[J]. Journal of Scientific Computing, 2019, 80(3): 1419-1437.



MAO Zhixiong, born in 2002, undergraduate, is a member of CCF (No. Q7701G). His main research interests include graph computation and storage.

WANG Mengxiang, born in 1991, Ph.D, research assistant, is a member of CCF (No. 72448M). Her main research interests include graph data mining, geographic information system (GIS) and international standardization.

(责任编辑:柯颖)