基于 MPI 的 OTP 三角形计数算法研究

龙昌庭[1]
LONG Changting

摘 要 在社交网络分析、推荐系统和聚类系数等大规模图分析问题中，计算图中三角形的数量是一项重要的任务。然而，当面临大量数据以及子图之间存在重复的三角形结构时，计算变得困难且具有挑战性。因此，图数据的分布式计算变成了研究热点。提出一种基于 OTP 三角形计数算法的 MPI 优化算法，OTP 算法是基于 MapReduce 框架的三角形计数算法，但在三角形数量的计算过程中，计算时间仍然过长。通过实验结果的分析，发现优化后的算法相较于现有算法，在计算时间上显著缩短 10 ～ 40 倍，特别是在处理非常大规模图时。这一优化进一步弥补了现有算法在执行时间性能方面的不足。

关键词 图划分；三角形计数；分布式计算；邻接表；MapReduce；消息传递

doi：10.3969/j.issn.1672-9528.2024.02.029

0 引言

近年来，随着互联网信息技术的迅速发展，各行各业产生了大量的数据。图这种特殊的数据结构，不仅可以存储数据本身，还可以存储数据之间的复杂关联关系，因此受到了学术界和工业界的广泛关注。图可以表示许多重要的数据，如社交网络、生物信息网络、交通网络等，分析图数据可以得到重要的洞见。学术界研究图挖掘算法，以发现图数据中的模式和规律。工业界利用图分析技术，获得重要的商业智能，改进决策。总之，图作为一种关键的数据结构，能表示现实世界的复杂关系，分析图数据可以获得重要知识，因此研究图数据具有重要意义，受到学术界和工业界的广泛重视。

三角形数量是图分析中的一个重要任务，它主要用于识别和计算图中存在的三角形结构。通过对三角形数量的研究，可以度量网络的聚类系数，即节点之间的紧密程度。聚类系数可以帮助理解网络的群体结构、信息传播和社交影响力等，三角形数量是研究和分析复杂网络的重要指标[1]。在异常检测[2]中，也可以通过计算图中的三角形数量和分布情况，可以发现与正常情况不符的异常模式，在推荐系统[3]中，通过

1. 贵州财经大学信息学院 贵州贵阳 550025

[12] MOLANES L E M, CAO R, KEILEGOM I V. Smoothed empirical likelihood confidence intervals for the relative distribution with left-truncated and fight-censored data[J]. Canadian journal of statistics, 2010,38(3): 453-473.

[13] 茆诗松 , 汤银才 . 贝叶斯统计 [M]. 北京 : 中国统计出版社 , 2012.

[14] 何朝兵 , 刘华文 . 左截断右删失数据下对数正态分布参数多变点的贝叶斯估计 [J]. 福州大学学报 (自然科学版), 2014, 42(4):507-513.

[15] 彭秋曦 . 左截断右删失数据下指数分布变点的 Bayes 估计 [D]. 重庆 : 重庆大学 ,2015.

[16] 何朝兵 , 刘华文 . 左截断右删失数据下二项分布参数多变点的贝叶斯估计 [J]. 华南师范大学学报 (自然科学版), 2014, 46(3):34-38.

[17] 何朝兵 , 刘华文 . 左截断右删失数据下几何分布参数多变点的贝叶斯估计 [J]. 重庆师范大学学报 (自然科学版), 2014, 31(4):100-105.

[18] 梅梦玲 .IIRCT 下某些指数型分布族参数变点的贝叶斯估计 [D]. 乌鲁木齐 : 新疆师范大学 ,2021.

[19] BRADLEY E, TREVOR H I, ROBERT T. Least Angle Regression[EB/OL]. (2004-06-23)[2023-10-06]. https://arxiv.org/abs/math/0406456.

【作者简介】

李清宇（1999—），女，贵州铜仁人，硕士，研究方向：统计模型与统计计算。

黄介武（1977—），通信作者（email: 846221886@qq.com），男，湖南沅江人，博士，教授，研究方向：统计模型与统计计算。

（收稿日期：2023-11-27）

分析用户之间的三角形关系，可以发现潜在的兴趣相似的用户群体等等，因此，计算三角形在图分析中具有重要的作用。

但是，图数据的规模通常比较大，用顺序算法难以计算图数据，所以，各种分布式工具开始用来计算图数据，MapReduce 因为其高度可扩展性、容错性、高效的并行效率而让其成为处理大数据的典型工具。OTP 算法 [4] 是基于 MapReduce 框架下的分布式算法 TTP 算法 [5]，OTP 算法采用了一种图分区加强策略来保证不产生冗余的三角形，从而避免通过图划分而产生的重复三角形的计算问题，相比之前的算法，OTP 算法拥有更好的时间性能和工作负载。相比于 MapReduce，MPI 可以更灵活地管理进程间的消息传递和同步操作，MPI 提供了一系列的通信和同步操作，使得编写并行程序更加灵活。

1 相关工作

在大规模图中，已经有了很多关于三角形计算的研究。在 MapReduce 框架中，Cochen 算法 [6] 是第一个针对三角形计算的 MapReduce 算法。该算法引入了一个概念叫做二路边，即边 (u, v) 和边 (u, w) 构成了一条二路边，然后通过寻找连接这两条二路边的边 (v, w) 来形成一个三角形。然而，Cochen 算法存在一些缺点，包括网络过载和增加计算时间的问题。

为了解决这些问题，Suri 等人 [7] 提出了 GP 算法。与朴素方法相比，GP 算法能够明显加速计算，并且适用于 MapReduce 环境。该算法的一个重要特点是它可以平衡每台机器上的可用内存和算法总可用内存之间的关系，同时保持总的工作量不变。然而，GP 算法仍然存在网络过载和增加计算时间的问题，并且如果三角形的顶点位于同一分区，仍然会进行冗余计算。

为了克服这些问题，Park 等人提出了 TTP 算法。该算法采用了一种新的思路来计算图中的三角形数量，通过将三角形分类为三种类型，并根据不同类型的三角形进行不同的处理，大大减少了冗余计算。然而，如果三角形的三个顶点都位于同一个分区，TTP 算法仍然会进行冗余计算。

最近，Sharafeldeen 等人提出了 OTP 和 ETTP 两种 MapReduce 算法，这两种算法避免了三角形的冗余计算问题。在时间性能上，它们比之前的 TTP 算法有了显著的提升。

在 Ghosh 等人 [8] 提出的论文中，证明了用消息传递接口（MPI）在图中实现三角形计数的分布式内存，与在 NERSC Cori 节点的 32 个处理器核心上的工作相比，拥有高达 90 倍的加速。

2 算法模型

2.1 图划分模型

给定图 $G=(V, E)$，$|V|=n$，$|E|=m$，给定正整数 ρ，通过图划分算法将 G 按顶点的方式划分成 ρ 个交集为空的顶点集 $V_1, V_2, \cdots, V_\rho$，得到 ρ 个子图 $G_i=(V_i, E_i)$。当一条边的两个顶点只属于一个顶点集时，定义其为内边，否则，定义其为交叉边，通过 $P(v)=i\ |v\in V_i$ 来获取顶点 i 所处的顶点集。具体公式为：

$$G(V,E)=\sum_{i=1}^{\rho}G_i(V_i,E_i) \tag{1}$$

$$\{P(v)=i|v\in V_i\} \tag{2}$$

对于图中的三角形，经过划分后可以得到三种类型的三角形，即三个顶点都处于同一个顶点集的三角形、一个顶点处于某一个顶点集，而另外两个顶点处于另外一个顶点集的三角形和三个顶点都处于不同顶点集的三角形，就计算三角形的问题而言，定义三种三角形为 Type-1、Type-2、Type-3。

Type-1：三个顶点都在同一个顶点集的三角形，如图 1 所示，顶点 {10, 11, 12} 所构成的三角形属于 Type-1 的三角形。

Type-2：三个顶点分布在两个顶点集的三角形，如图 1 所示，顶点 {2, 3, 4} 所构成的三角形属于 Type-2 的三角形。

Type-3：三个顶点分布在三个顶点集的三角形，如图 1 所示，顶点 {3, 4, 10} 所构成的三角形属于 Type-3 的三角形。

如图 1 所示，图 G 分成四个顶点集，分别是顶点 {1, 2, 3}、{4, 5, 6}、{7, 8, 9}、{10, 11, 12}，可以看到，通过划分之后，不同顶点集之间存在边关系，如何处理交叉边，成为避免冗余三角形计算的关键。

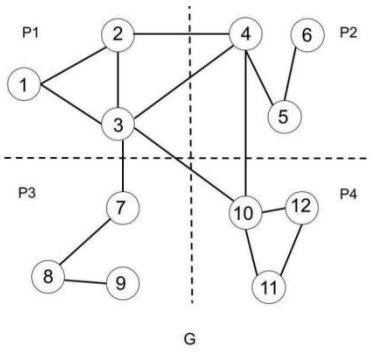

图 1 示例图 G

在第一个顶点集中，其他顶点集的顶点 u 如果和第一个顶点集的 n 个顶点有 $u_j(j=1,2,\cdots,n)$ 边，就把顶点 u 和这 n 个顶点一一构成的边添加到第一个顶点集所构成的图 G_1 中。

在图 1 中，属于第一个分区的顶点 {1, 2, 3} 有这 3 个顶点，对于顶点 2 来说，它和顶点 4 存在一条边 (2,4)，对于顶点 3 来说，它和顶点 4 存在一条边 (3,4)，和顶点 7 存在一条边 (3,7)，和顶点 10 存在一条边 (3,10)，和这三个顶点有关系的边就是 (2,4)、(3,4)、(3,7)、(3,10) 这 4 条边，把这 4 条边都添加到原来第一个顶点集所构成的图 G_1 当中，就形成新的顶点集和新的图 G_1。

此时，$P(1)=P(2)=P(3)=P(4)=1$，可以发现，原来处于第 2 个分区的顶点 4 添加到 G_1 之后，三角形 (2,3,4) 由 Type-2 的

三角形变成了 Type-1 的三角形。

在这种分区模型中，不用担心会产生重复的三角形从而导致冗余的三角形计算，因为添加边的时候，所有 G_i 分区的情况其实只有两种情况，一是包含三角形的一个顶点，如 G_2 里面就包含了三角形 (2,3,4) 的顶点 {4}，另一种情况就是包含三角形的两个顶点，如 G_1 中包含了三角形 (2,3,4) 的两个顶点 {2,3}。

不产生重复三角形的关键就是为什么两个分区都添加边之后却不形成重复的三角形，对于 G_1 来说，如果三角形有两个顶点处于 G_1，那么就只剩下一个顶点处于 G_2 或者其他分区。

用三角形 (2,3,4) 举例说明，G_1 中包含了三角形 (2,3,4) 的两个顶点 {2,3}，也就是边 (2,3)，由图 1 可知，顶点 4 与顶点 {2,3} 都存在边关系，因此添加边 (2,4) 和边 (3,4)，添加了另外两条边 (2,4) 和 (3,4) 之后，就构造了图 1G 中的三角形 (2,3,4)。

而对于 G_2 来说，只包含一个顶点 4，把边 (2,4) 和边 (3,4) 添加到 G_2 中以后，顶点 {2,3,4} 只有两条边，无法构成三角形，因为添加边的行为只是针对外部的顶点和当前分区内的顶点存在边的情况才进行，外部顶点之间的边不添加到当前分区。所以，如果只有一个顶点在某一个分区，那么这一个分区即使添加边，也无法形成三角形，保证了不产生冗余的三角形。

此时，已经把 Type-2 的三角形转换成了 Type-1 的三角形，剩下 Type-3 的三角形，在之前的处理中，因为已经把 Type-2 的三角形转换成了 Type-1 的三角形，也就是说，顶点在同一个分区和处于两个分区的三角形已经被正确计算，只剩下顶点处于三个分区的三角形没有被计算。

对于这些三角形的处理则是只添加交叉边到新的分区 G_{ijk} 里，当交叉边的分区属于 $\{i,j,k\}$ 当中时，把交叉边添加到 G_{ijk}，$\{P(u) \neq P(v) \neq P(w) | (P(u),P(v),P(w)) \in (i,j,k)\}$，其中，$1 \leq i < j < k \leq \rho$。

当 i=1、j=2、k=4 时，G_1 和 G_2 中的交叉边是 (2,4)、(3,4)，G_1 和 G_2 的交叉边是 (3,10)，G_2 和 G_4 的交叉边是 (4,10)，把所有边添加到 G_{124} 可以得到三角形 (3,4,10)。结合图 1 可知，至此所有类型的三角形都被正确的添加到不同类型的子图里。由于没有重复的三角形，所以保证了每个三角形只会被计算一次，由此解决了重复三角形的计算问题。

2.2 改进算法模型

OTP 算法中，在对三角形计算时，首先得到两个顶点的邻居，然后对邻居排序，把两个顶点的邻居列表元素进行比较，邻居不相等的情况下，较小的邻居元素往前移动，直到找完所有相等的邻居。因涉及到邻居列表的排序操作，可能会对性能产生一定的影响。因此，本文采用了邻接表表示法来处理图数据，邻接表表示法提供紧凑的方法来表示稀疏图（通常指 $|E|$ 远小于 $|V^2|$，其中 $|E|$ 指图的边的数量 $|V|$ 指图的顶点的数量。

因为实际的图数据中，大多数图数据都是稀疏图，这是由于图数据的特性和实际应用而决定的。大多数图数据都是稀疏图，这是由于图数据的特性和实际应用而决定的。其次，稀疏图的存储和处理效率更高。

由于稀疏图中有较少的边，因此可以节省存储空间，并且在处理图算法时可以减少计算量。相比之下，如果图是密集图，每个节点都与其他节点直接相连，那么存储和处理这种图将需要更多的资源和时间。此外，对于大规模图数据，存储和处理整个图可能会变得非常困难，甚至不可行。稀疏图可以更好地适应分布式计算框架和并行算法，因为可以通过分片和分区来处理图数据，使得处理过程更加高效和可扩展。因此，使用邻接表可以更好地利用存储和计算资源，并且适合于处理大规模的图数据。在改进的算法中，在 Map 阶段的处理中，首先加载图数据集然后根据 ρ 的值对图进行划分，然后循环遍历边集。在一开始的分区 G_i 中，每一个分区都包含了分区内所有顶点的边，所以读取到边之后，根据图划分的逻辑，如果一条边 (u,v) 的两个顶点都处于同一个分区，那么则直接把这条边划分到这两个顶点所处的分区，此时发送分区信息 $\langle P(u); (u,v) \rangle$ 给 Reduce 阶段。其他分区的顶点与当前分区内的顶点如果存在边关系，那么就把这条边 (u,v) 添加到两个顶点所对应的分区，此时把分区信息 $\langle P(u); (u,v) \rangle$、$\langle P(v); (u,v) \rangle$ 发送给 Reduce 阶段。

通过这样的处理就得到了相同分区内的所有 Type-1 和经过处理的 Type-2 三角形。例如对于 G_1 来说，读取到边 (1,2) 之后，就把边 (1,2) 添加到顶点 1 所处的分区中，此时发送 1 所在的分区 $P(1)$ 以及这条边 (1,2) 给 Reduce 阶段，经过遍历之后，分区 1 发送到 Reduce 阶段的键值就是 <1; (1,2),(1,3),(2,3),(2,4),(3,7),(3,10) >，其中，1 作为键，其他边作为值。所有边都被完整发送到 Reduce 之后，加下来就是对 G_{ijk} 进行处理，G_{ijk} 只包含交叉边，所以一条边只要是交叉边且所属的两个分区号是 $\{i,j,k\}$ 的一个子集，那么就把这边添加到 G_{ijk}，对于 G_{124} 来说，边 (3,4) 属于分区 1、2，而分区 1、2 是 {1,2,4} 的一个子集，此时把 {1,2,4} 和边 (3,4) 发送到 Reduce 阶段，经过这样的处理，就把 G_{124} 的子图发送给了 Reduce 阶段，对于所有的边都进行这样的处理之后，就把这一类型的子图发送给了 Reduce 阶段。

在 Hadoop 的框架中，Shuffle 阶段是对用户隐藏的，Shuffle 阶段会自动对 Map 阶段发送的键值对进行排序，之后再把排序后的键值对发送给 Reduce 阶段。基于 Shuffle 阶段的处理，本文改进的 MPI 算法中，采用了 C++ 里的关联容器 map，用于存储键值对。基于红黑树实现，向 std::map

插入新的键值对时，它会自动根据键的值进行排序，以保持有序状态。这使得在 std::map 中进行查找、插入和删除操作时都能够以较快的速度进行。首先通过把所有的键值对储存到 map 容器中，Reduce 读取依次读取键值并把键值还原对应的分区和分区内的边，此时所有键值都被处理成一个子图的形式，如 <1; (1,2),(1,3),(2,3),(2,4),(3,7),(3,10) > 和 <(1,2,4); (2,4),(3,4),(3,10),(4,10)> 对应了 G_1 子图和 G_{124} 子图，其中，键 1 和键 (1,2,4) 为子图的一个标识。然后 Reduce 开始处理每一个子图，在改进后的 MPI 算法中，采用邻接表对图数据进行处理，处理每一条边的两个顶点，通过遍历两个顶点的邻居，可以快速找到相同的邻居，如果找到相同的邻居，则找到了一个三角形，此时输出三角形。

3 实验与分析

3.1 实验数据与环境

本文实验数据集全部来自斯坦福大学公开的数据集 Stanford network analysis project（SNAP）[9-10]，数据集的详细信息如表 1 所示。

表 1 数据集基本信息

数据集	节点	边	三角形数量
ego-Facebook	4039	88 234	1 612 010
Brightkite	58 228	428 156	494 728
AS-733	6474	13 895	6584
ca-AstroPh	18 772	396 160	1 351 441
CA-CondMat	23 133	186 936	173 361
CA-HepTh	12 008	237 010	3 358 499
Email-Enron	36 692	367 662	727 044
p2p-Gnutella08	6301	20 777	2383
p2p-Gnutella31	62 586	147 892	2024
soc-Epinions	75 879	508 837	1 624 481
wiki-Vote	7115	103 689	608 389

其中，数据集格式为每行两个顶点，所需的数据集为无向图且不包含重复的边，通过数据预处理去重可以得到需要的格式。实验环境是一个拥有 80 个节点的计算集群，每个节点的 CPU 为 Intel Xeon E5-2630 v4 @2.20GHz，运行内存为 128 GB。

3.2 实验结果

在实验中，设置分区数量为 20、进程数为 15 分别对每个数据集分别用改进后的 MPI 算法、OTP 算法和 ETTP 算法进行测试，两个算法都是最近提出算法且在时间性能上都优于以往的算法，测试结果如表 2 所示。

表 2 数据集及各算法运行时间

数据集	MPI/s	OTP/s	ETTP/s
ego-Facebook	3.7	38.5	25.5
Brightkite	7.9	315.2	146.5
AS-733	0.4	2.4	2.5
ca-AstroPh	5.6	207.0	63.5
CA-CondMat	1.9	43.85	16.3
CA-HepTh	0.5	3.9	2.8
Email-Enron	11.8	253.5	164.8
p2p-Gnutella08	0.4	3.0	2.5
p2p-Gnutella31	3.1	137	50.8
soc-Epinions	67.0	2056	2 637.6
wiki-Vote	5.6	85	39.9

由实验结果可知，改进后的 MPI 算法在时间性能上 OTP 算法和 ETTP 算法快了 10 ～ 40 倍。为了进一步证明算法的稳健性，分别把 MPI 使用的进程数从 1 ～ 20 分别在三个算法上运行了 soc-Epinions 数据集和 ego-Facebook 数据集，以及在 p2p-Gnutella31 数据集和 Email-Enron 数据集上测试当分区数取不同的值时对三个影响。实验结果如图 2 ～图 5 所示。

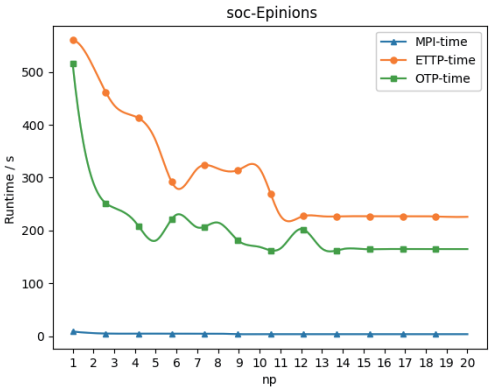

图 2 soc-Epinions 数据集不同进程数时间对比图

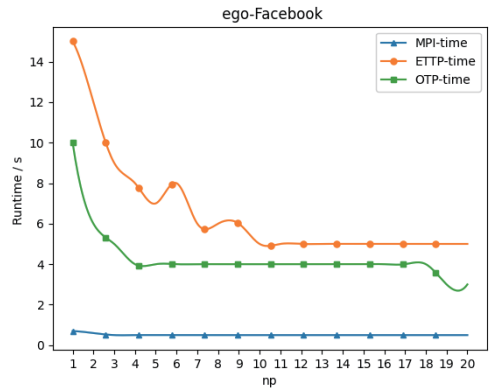

图 3 ego-Facebook 数据集不同进程数时间对比图

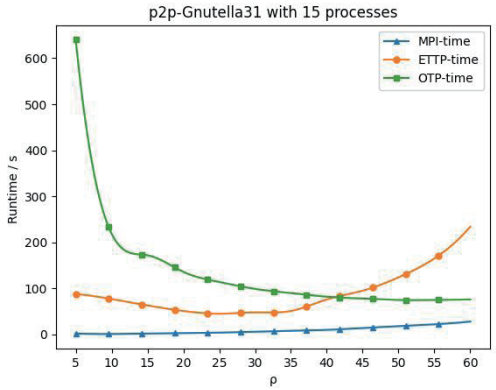

图 4 不同值在 p2p-Gnutella31 数据集的运行时间

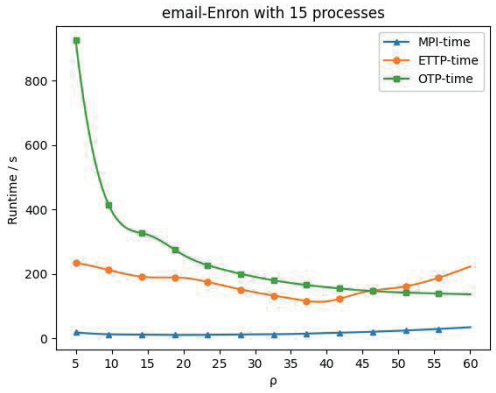

图 5 不同值在 email-Enron 数据集的运行时间

可以看到，改进后的算法在 soc-Epinions 数据集的运行时间上，远比 OTP 算法和 ETTP 算法所需要的时间少。虽然在具体的时间上，改进后的算法随着进程数量的增多，速度最高只提升了一倍，而 OTP 算法和 ETTP 算法最快能提升 3 倍，但是改进后的算法仍然比 OTP 算法和 ETTP 算法要少，为了进一步验证，在 ego-Facebook 数据集再进行测试，结果如图 3 所示。由图 3 可以看出，运行时间正如图 2 分析的那样，这证明了在不同的数据集算法的效率是一致的。为了进一步验证数据在不同值上的性能，让三个算法在 p2p-Gnutella31 和 email-Enron 数据集上进行不同值的运行时间对比，结果如图 4 和图 5 所示。

由图 2 ～图 5 的实验结果可以证明，不论是随着 MPI 进程数的增加还是随着分区数的增加，改进的 MPI 算法的时间都远低于 OTP 算法和 ETTP 算法。

4 结论

本文使用了消息传递接口（MPI）实现并优化了 OTP 算法。通过对斯坦福大学公开的多个图数据集进行实验，证明了优化后的算法在时间性能上相比于传统的 OTP 算法和 ETTP 算法有着显著的提升。实验结果显示，优化后的算法的执行时间是 OTP 算法和 ETTP 算法 10 ～ 40 倍。实验结果进一步表明，随着数据集的规模增大，优化算法的性能提升速度也随之加快。换言之，当处理大规模数据集时，优化算法的效果尤为明显。这一发现对于处理大型图数据集的实际应用具有重要意义，因为大型图数据集在现代应用中越来越常见。

参考文献：

[1] TSOURAKAKIS C E, PACHOCKI J, MITZENMACHER M. Scalable motif-aware graph clustering[EB/OL].(2017-04-03)[2023-09-28].https://arxiv.org/abs/1606.06235.

[2] FORTUNATO S. Community detection in graphs[J]. Physics reports, 2010, 486(5): 75-174.

[3] STRAUSZ A, VELLA F, DI GIROLAMO S, et al. Asynchronous distributed-memory triangle counting and lcc with rma caching[EB/OL].(2022-05-30)[2023-09-20].https://ieeexplore.ieee.org/document/9820724/.

[4] SHARAFELDEEN A, ALRAHMAWY M, ELMOUGY S. Graph partitioning mapreduce-based algorithms for counting triangles in large-scale graphs[J]. Scientific reports, 2023, 13(1): 166-180.

[5] PARK H M, CHUNG C W. An efficient mapreduce algorithm for counting triangles in a very large graph[C]//In Proceedings of the 22nd ACM international conference on Information \& Knowledge Management.New York:ACM,2013: 539-548.

[6] COHEN J. Graph twiddling in a mapreduce world[J]. Computing in science & engineering, 2009, 11(4): 29-41.

[7] SURI S, VASSILVITSKII S. Counting triangles and the curse of the last reducer[C]//In Proceedings of the 20th international conference on World wide web.New York:ACM,2011: 607-614.

[8] GHOSH S, HALAPPANAVAR M. Tric: Distributed-memory triangle counting by exploiting the graph structure[C]//2020 IEEE high performance extreme computing conference (HPEC). Piscataway:IEEE, 2020: 1-6.

[9] JURE L,ANDREJ K. SNAP datasets: stanford large network dataset collection[EB/OL].[2023-09-12]. http://snap.stanford.edu/data.

[10] DEAN J, GHEMAWAT S. Mapreduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

【作者简介】

龙昌庭（1996—），男，贵州毕节人，硕士研究生，研究方向：图计算、图划分。

（收稿日期：2023-11-24）