计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(3):322−329 [doi: 10.15888/j.cnki.csa.009032]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 NUMA 延迟发送的时变图弱连通分量求解①

梁锐杰, 程永利

(福州大学 计算机与大数据学院/软件学院, 福州 350116)
通信作者: 程永利, E-mail: chengyongli@fzu.edu.cn

摘　要: 时变图连通分量已经被广泛应用到不同场景, 如交通路网建设、推荐系统的信息推送等. 然而当前多数连通分量求解方法忽视了 NUMA 体系结构对计算效率产生的影响, 即过高的远程内存访问延迟导致低下的算法执行效率. 本文针对时变图的弱连通分量求解问题, 提出一种基于 NUMA 延迟发送的时变图弱连通分量求解方法, 它通过合理的数据内存布局, 合理控制 NUMA 节点间的信息交换次数, 最大限度减少远程内存访问数量, 显著提高了算法执行效率. 实验结果表明, 该方法的性能明显优于当前流行的图处理系统 Ligra 和 Polymer 提供的方法.

关键词: 弱连通分量; NUMA; 延迟发送; 时变图; 图计算

引用格式: 梁锐杰,程永利.基于 NUMA 延迟发送的时变图弱连通分量求解.计算机系统应用,2023,32(3):322–329. http://www.c-s-a.org.cn/1003-3254/9032.html

NUMA-based Delayed Sending Method for Weakly Connected Components of Time-evolving Graph

LIANG Rui-Jie, CHENG Yong-Li

(College of Computer and Data Science/College of Software, Fuzhou University, Fuzhou 350116, China)

Abstract: The weakly connected components of the time-evolving graph have been widely used in many areas, such as traffic network construction, information push of recommendation systems, etc. However, most methods for the weakly connected components ignore the impact of the non-uniform memory access (NUMA) architecture, that is, the high remote memory access delay leads to low execution efficiency. This study proposes a NUMA-based delayed sending method to find the weakly connected components of the time-evolving graph. It minimises the number of remote accesses and improves computational efficiency through reasonable data memory layout and controlling the number of exchanges between NUMA nodes. The experimental results show that the performance of the NUMA-based delayed sending method is better than the methods provided by the current popular graph processing systems Ligra and Polymer.

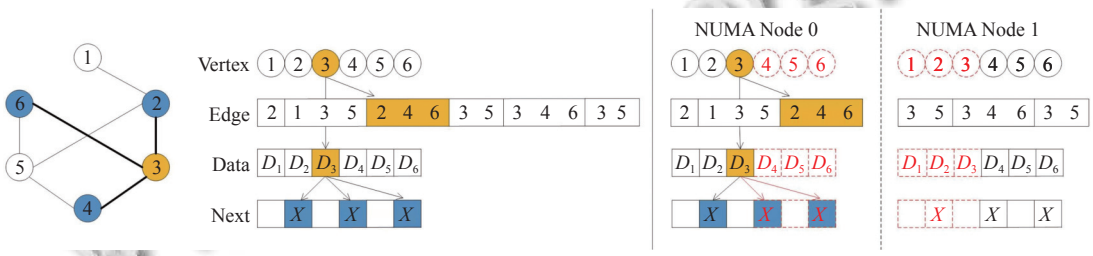
Key words: weakly connected components; non-uniform memory access (NUMA); delayed sending; time-evolving graph; graph computing

现实世界无时无刻不在改变, 许多现实世界的问题可以用时变图建模、求解. 时变图 (time-evolving graph, *TEG*) 由一系列在时间上连续的快照组成, 可表示为 $TEG=\{S_i\}$, 其中 i 是从 0 开始递增的整数. 每张快照 $S_i=(V, E)$ 表示图结构在某个时间点的状态, 其中 V 为 S_i 的顶点集, E 为 S_i 的有向边集. 时变图计算任务通常可分为两个阶段[1]: (1) 在多张历史快照上运行相同的静态图算法; (2) 根据历史快照的计算结果研究现实世界的发展规律, 预测未来发展趋势, 为国家安全、金融、新零售、社交、政府、企业等重要领域提供决策支持[2,3]. 由于存在巨大的潜在产业价值, 时变图计算成为当前学术界和工业界共同关注的研究热点[4].

① 基金项目: 福建省自然科学基金 (2020J01493)
收稿时间: 2022-08-23; 修改时间: 2022-09-27; 采用时间: 2022-10-21; csa 在线出版时间: 2022-12-23
CNKI 网络首发时间: 2022-12-27

时变图连通分量挖掘首先在多个历史快照上分别求解连通分量, 然后根据不同快照的求解结果研究现实世界的发展规律, 具有重要的研究意义和应用价值. 例如, 在推荐系统中, 通过分析各快照连通分量的变化, 可以衡量在线零售行业的用户变化情况, 以此提高推荐系统的推荐质量和准确性[5]. 在机会网络上, 时变图的连通分量被用来预测网络拓扑结构的变化[6] .在交通路网中, 时变图连通分量的变化被用来衡量各时间段的路网拥堵状况, 为智能交通管控提供决策支持[7].

标签传播算法是当前流行的连通分量求解方法[8], 其主要思想如下: 各顶点在计算开始时被赋予唯一的标签. 整个算法执行过程由多轮迭代组成. 当执行每轮迭代时, 所有顶点并行执行, 每个顶点把各自的标签传播给邻居, 而邻居则需要比较接收的标签, 从中选择最小的标签作为下一轮迭代的初始值. 经过多轮迭代, 当所有顶点的标签不再发生改变, 则计算结束, 具有相同标签的顶点同属一个连通分量. 如图 1(a) 所示, 顶点 3 向顶点 2, 4, 6 传播标签, 而顶点 2, 4, 6 会在各邻居的标签中选择最小的标签作为下一轮迭代的初始值.



(a) 标签传播算法 (b) NUMA 体系架构的内存布局

图 1 标签传播算法及其在 NUMA 体系架构的内存布局

该算法在每轮迭代过程中, 所有顶点都需要访问它的所有邻居, 产生大量随机内存访问, 很大程度上降低了算法的执行速度. 主要原因如下: non-uniform memory access (NUMA) 体系架构是当前流行的计算机体系结构[9]. NUMA 体系架构的计算机包含多个 NUMA 节点, 各个节点由多个核和一块局部内存组成. NUMA 节点间通过总线互相连接形成全局共享内存. 然而, 该内存分布结构导致了非统一内存访问, 即访问远程 NUMA 节点内存的延迟要明显高于访问本地节点内存的延迟[10]. 在 NUMA 体系架构的计算机上执行标签传播算法, 顶点和边会被随机分配在不同的 NUMA 节点. 顶点在计算执行过程需要不断向邻居传播标签, 而多数邻居分布在不同的 NUMA 节点, 由此产生远程随机访问. 如图 1(b) 所示, 6 个顶点恰好被随机均匀地分配在 2 个 NUMA 节点. 其中, 顶点 3 和邻居 2 同在 NUMA 节点 0 的内存, 而邻居 4 和 6 则在另一节点的内存, 执行线程需要通过连接总线才能向顶点 4 和 6 传播标签, 由此产生了 2 次远程随机访问.

为此, 本文围绕标签传播算法在 NUMA 体系架构上由于过多的远程内存访问导致时变图连通分量求解效率低问题, 提出了基于 NUMA 感知的延迟发送方法. 通过设计延迟发送策略, 将多轮迭代的信息进行合并发送, 即多轮迭代才完成一次信息交换, 明显减少节点间的信息交换次数, 从而大幅提升算法的执行速度. 同时, 该方法还通过合理的内存数据布局设计, 尽可能降低远程访问数量, 进一步提升算法的执行效率. 实验结果表明, 该方法在保证计算结果正确性的同时, 可获得显著的性能提升, 与流行的图处理系统 Ligra[11] 和 Polymer[12] 提供的方法相比, 性能提升达 1.5–2.5 倍.

1 基于 NUMA 感知的延迟发送方法

1.1 总体概述

基于 NUMA 感知的延迟发送方法首先根据 NUMA 体系架构内存访问特性, 合理分配图数据到各个 NUMA 节点, 构建基于 NUMA 感知的时变图数据内存布局. 接着, 各个 NUMA 节点进行局部计算, 并根据计算状态延迟 NUMA 节点间的信息交换. 当各个 NUMA 节点内的计算都处于收敛状态时, 对 NUMA 节点进行全局同步: 选择半数的活动 NUMA 节点发送本地标签块给另外半数的活动 NUMA 节点, 其中, 发送了标签块的 NUMA 节点立刻被灭活, 不再参与后续计算; 而其他活动的 NUMA 节点接收到标签块后则进行两次扫

描更新本地标签块. 算法迭代地进行 NUMA 节点间的全局同步, 直到只剩一个活动的 NUMA 节点的本地标签块被更新, 计算结束, 具有相同标签的顶点同属一个连通分量.

1.2 NUMA 感知的时变图数据内存布局

为了降低 NUMA 体系架构对图计算效率的影响, 节省内存开销, 提高计算速度, 我们设计了一种 NUMA 感知的时变图数据内存布局. 该项设计由两部分组成, 即 NUMA 感知的数据内存布和时变图数据内存布局.

1.2.1 NUMA 感知的数据内存布局

NUMA 感知的数据内存布局把图数据分为图拓扑数据与顶点标签数据. 对于图拓扑数据, 例如顶点和边, 各访问线程能够在各自的 NUMA 节点申请内存空间存储各自能够访问的图拓扑数据. 而对于顶点标签数据, 我们允许各个 NUMA 节点存储其他 NUMA 节点的标签副本.

该数据内存布局尽可能地降低了远程随机访问的数量. 首先, 访问线程不需要访问其他的图拓扑数据, 只需要访问它们持有的图拓扑数据. 因此, 把相关的图拓扑数据绑定访问线程所在的 NUMA 节点, 能够消除对图拓扑数据的远程访问. 接着, 顶点标签数据在计算过程需要被频繁交换和更新, 不可避免地产生远程随机访问. 针对该问题, 我们通过在本地节点上创建远程节点的标签副本. 为此, 算法局部计算过程中, 数据的更新只需在本地的 NUMA 节点进行, 减少了顶点标签数据的远程随机访问. 只有在全局数据交换过程, 才需要完成节点间的标签数据交换. 实验表明, 标签副本的内存开销很小.

1.2.2 时变图数据内存布局

尽管对各种图数据进行了合理分配, 尽可能降低了远程随机访问的数量, 但是若忽略了时变图的内存布局设计, 将会导致巨大的内存开销. 时变图由多张在时间上连续的快照组成, 快照间通常有大量重叠的顶点和边, 具有高度相似性[13]. 若为所有快照构建数据内存布局, 势必造成不必要的内存浪费. 为此, 我们设计了时变图的位图表示结构: 将所有快照的顶点集合作并集运算, 避免了不同快照间顶点的重复存储. 相似地, 所有快照的边集合作并集运算, 避免了不同快照间边的重复存储. 同时, 为并集的每个顶点或边设计一个位图 (bitmap), 一个位图共有 N 位, N 为快照的数量; 每一位的 1 或 0 表示该顶点或边是否属于某一个快照. 如图 2(a) 所示, 边的位图由 2 个 bit 组成, 表示快照数量为 2. 其中, 边 (v_1, v_2) 的位图为 10, 表示该边存在于快照 S_1, 而不属于快照 S_2.

此外, 顶点标签数据的结构设计同样会对计算效率产生影响. 顶点标签数据在时变图上有两种设计思路[14], 一种是时间局部性设计, 一种是空间局部性设计. 时间局部性设计如图 2(b) 所示, 不同快照同一顶点的标签放在相邻位置. 该设计能够顺序地读取顶点在不同快照的标签, 同时更新多张快照. 而空间局部性设计如图 2(c) 所示, 顶点标签依据快照次序顺序地放在相邻位置. 该设计能够顺序地处理各张快照. 尽管空间局部性设计有直观的计算过程, 但其会造成比时间局部性设计更多的 cache 未命中. 如图 2(b)、图 2(c) 所示, 假设 cache line 大小为 2, 若顶点 v_1 向 v_3 传播消息, 而 cache line 没有读取到 v_3 的标签则造成一次 cache 未命中. 可以看见, 采用时间局部性设计同时处理所有快照只造成 1 次 cache 未命中, 而采用空间局部性设计顺序处理各快照造成 2 次 cache 未命中. 因此, 为了提高时变图连通分量的求解效率, 我们采用时间局部性设计.

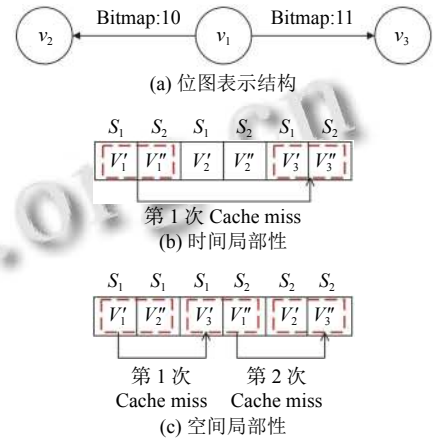


图 2 位图表示结构及顶点标签的两种设计思路

当完成时变图所有快照的预处理, NUMA 感知的时变图数据内存布局如图 3(a) 所示. 每个 NUMA 节点都存储着相等数量的顶点, 各顶点的出度边都有对应的位图. 此外, NUMA 节点除了存储自己顶点的标签外, 还存储其他 NUMA 节点的顶点标签副本, 并且它们均采用时间局部性设计. 为了方便区分不同快照的顶点标签, 图中使用黑色来表示快照 1 的顶点标签, 使用红色来表示快照 2 的顶点标签.

1.3 延迟发送策略

当完成 NUMA 感知的时变图数据内存布局的构建, 就能够使用延迟发送策略在该内存布局上快速求解所有图快照的连通分量. 延迟发送策略把计算划分为局部计算和全局同步两个主要阶段. 在局部计算阶段, 各 NUMA 节点并行地求解其内部所有快照子图的连通分量. 当所有 NUMA 节点的计算状态都处于局部收敛, 则进入全局同步阶段. 全局同步阶段包括信息发送阶段和两次扫描阶段, 其主要思想如下: 把所有活动的 NUMA 节点分成数量相等的两份, 并选择其中一份向另一份发送标签块, 其中, 发送了标签块的 NUMA 节点立刻被灭活, 不再参与后续计算; 而活动的 NUMA 节点则进行两次扫描. 第 1 次扫描根据本地标签块和远程标签块的差异构建并查集, 第 2 次扫描则依照各标签在并查集上对应的根标签更新本地标签块. 算法迭代地进行全局同步阶段直到只剩一个活动的 NUMA 节点, 计算结束. 图 3 用一个实例演示了延迟发送策略的具体过程.

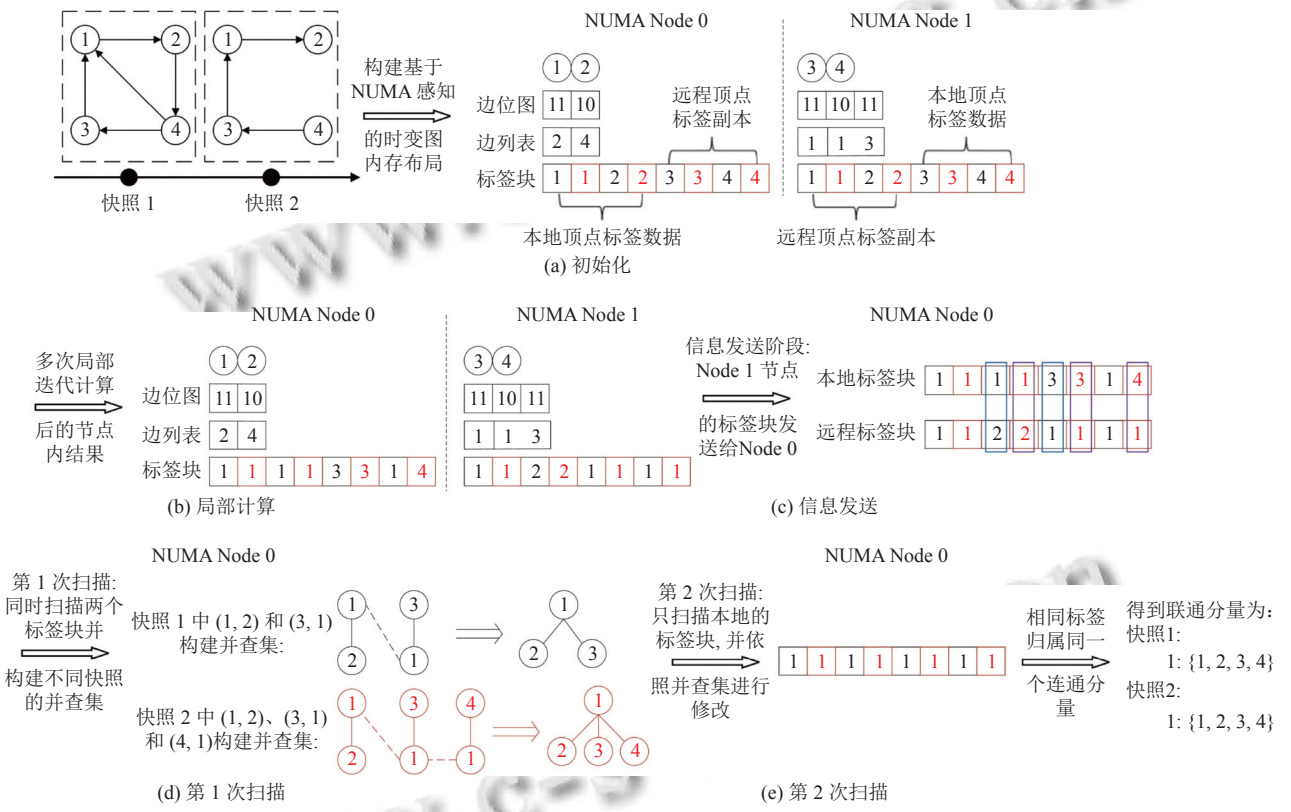


图 3 延迟发送策略实例

延迟发送策略的具体步骤如下.

① 初始化阶段: 初始化所有顶点在所有快照的标签, 使每个顶点在每张快照都有唯一的标签. 如图 3(a) 的标签块所示.

② 局部计算阶段: 该阶段的计算方式与标签传播算法类似. 各顶点并行地访问和比较出度邻居的标签. 若邻居的标签较大, 则用自己的标签更新邻居, 反之则用邻居的标签更新自己. 标签的更新只发生在 NUMA 节点内部的标签块, 并且更新的前提是两者都处于同一快照, 这可以通过边位图判断. 以图 3(b) 的顶点 2 为例, 顶点 2 访问邻居 4 在 NUMA 节点 0 的本地标签副本, 由于边 (2, 4) 的位图是 10, 顶点 2 与 4 的更新只能发生在快照 1, 而无法发生在快照 2. 图 3(b) 的标签块为多轮局部计算的结果.

③ 当所有 NUMA 节点完成一轮局部计算, 主线程检查本轮计算是否发生了标签更新. 若是, 则让发生标签更新的 NUMA 节点继续进行步骤②和步骤③; 若否, 则进入全局同步阶段.

④ 信息发送阶段: 把所有活动的 NUMA 节点分成数量相等的两份, 并选择其中一份向另一份发送本

地标签块, 其中, 发送了标签块的 NUMA 节点立刻被灭活, 不再参与后续计算. 如图 3(c) 所示, NUMA 节点 1 向 NUMA 节点 0 发送了本地标签块, 并且 NUMA 节点 1 不再参与后续计算, 其中, NUMA 节点 1 的标签块在 NUMA 节点 0 内被称为远程标签块.

⑤ 两次扫描阶段: 所有活动的 NUMA 节点并行地进行两次扫描. 在第 1 次扫描, 比较本地标签块与远程标签块相同位置的标签是否相等; 若否, 查找两个标签对应的根标签 (全局同步阶段开始时默认各标签对应的根标签是自己), 并记录查找过程的中间标签. 若两者的根标签不相等, 则修改大者的根标签, 使其指向小者的根标签. 不管两者的根标签是否相等, 都会让中间标签指向最小的根标签; 如图 3(d) 所示, 第 1 次扫描能够找到 (1, 2), (1, 2), (3, 1), (3, 1) 和 (4, 1) 这 5 组标签对, 根据标签对在标签块的位置计算出它们所属的快照, 从而构建出不同快照的并查集. 例如, 第 2 个位置的 (1, 2) 和第 4 个位置的 (3, 1) 都属于快照 1 的标签对, 在构建并查集时, 首先查找标签 1 和 2 对应的根标签, 因为全局同步阶段开始时默认各标签都是自己对应的根标签, 因此比较后让根标签 2 指向 1. 若存在中间标签, 同样使中间标签指向 1. 接着查找和比较标签 1 和 3 对应的根标签, 假设标签 3 对应的根标签比标签 1 的根标签小, 那么修改标签 1 的根标签, 让其指向标签 3 的根标签. 当然, 这里标签 1 和 3 的根标签都为自己, 因此修改标签 3, 让其指向标签 1. 第 1 次扫描结束, 完成了所有快照并查集的构建. 接着对本地标签块进行第 2 次扫描, 根据扫描位置计算出该位置所属的快照, 在对应快照的并查集查找该标签对应的根标签, 并记录查找过程的中间标签. 找到根标签后, 让中间标签指向根标签并更新标签块. 如图 3(e) 所示, 标签块内的第 4 个位置的标签是 3, 根据位置 4 计算出该标签属于快照 1, 因此在快照 1 的并查集查找标签 3 的根标签, 并修改标签块内该位置的标签为 1.

⑥ 当所有活动的 NUMA 节点完成 2 次扫描, 主线程检查是否只剩余一个活动的 NUMA 节点. 若否, 迭代进行步骤④–⑥; 若是, 则计算结束; 如图 3(e) 所示, 两次扫描结束后, 主线程发现当前只有一个活动的 NUMA 节点 0, 因此结束计算.

⑦ 分类: 按照各顶点的标签进行分类, 注意这时同样需要根据标签的位置计算出标签所属的快照, 具有相同标签的顶点同属一个连通分量.

1.4 延迟发送策略的时间复杂度分析

延迟发送策略把多轮迭代的信息进行合并发送, 明显减少节点间的信息交换次数. 因此, 对延迟发送策略的复杂度分析主要包括两个方面: 局部计算阶段的时间复杂度分析以及全局同步阶段的时间复杂度分析, 如表 1 所示.

在局部计算阶段, NUMA 节点内子图的顶点在每次迭代都需要与邻居进行比较, 因此, 一次局部计算的平均时间复杂度为 $O(|V|+|E|)$, 其中, $|V|$, $|E|$分别为子图的顶点数量和边数量. 而迭代次数只与 NUMA 节点内子图的直径 d 有关, 因此, 局部计算阶段的平均时间复杂度为 $O(d(|V|+|E|))$.

全局同步阶段的执行次数与 NUMA 节点数量 n 有关, 每次信息发送后, 半数的 NUMA 节点不再参与后续计算, 因此, 全局同步阶段的执行次数为 $\log n$ 次. 在两次扫描阶段, 第 1 次扫描并比较两组顶点标签块相同位置的标签对应的根标签, 同时对中间标签进行路径压缩, 因此其平均时间复杂度 $O(\alpha(|V|))$, α 表示阿克曼函数的反函数[15]. 第 2 次扫描则根据并查集修改顶点标签, 并且查询过程同样进行路径压缩, 故第 2 次扫描的平均时间复杂度为 $O(\alpha(|V|))$. 综上所述, 全局同步阶段的平均时间复杂度为 $O(2\alpha(|V|)\log n)$.

表 1 延迟发送策略的时间复杂度

阶段	时间复杂度
局部计算阶段	$O(d(\|V\|+\|E\|))$
全局同步阶段	$O(2\alpha(\|V\|)\log n)$

2 实验分析

2.1 实验环境

实验在 2 个 NUMA 节点构成的服务器进行, 硬件环境如表 2 所示.

表 2 服务器硬件环境

Node	CPU	Num of cores	Memory (GB)
NUMA Node 0	E5-2650 v4@ 2.20 GHz	24	128
NUMA Node 1	E5-2650 v4@ 2.20 GHz	24	128

2.2 实验对比对象

Ligra 和 Polymer 是当前图计算领域流行的图处理系统, Ligra 根据活动顶点数量在自上而下的 BFS 和自下而上的 BFS 间进行自适应切换, 从而减少无效顶点的随机访问. 尽管 Ligra 精心设计了自适应的访问模

式, 但是 Polymer 发现其在 NUMA 体系架构仍然产生大量的远程随机访问. 因此, Polymer 在 Ligra 的基础上, 针对图的拓扑数据、顶点计算数据以及运行状态数据的不同特性, 分别将它们合理安排在不同的 NUMA 节点与连续的虚拟地址空间, 以减少远程随机访问的数量.

2.3 数据集

Flickr 是雅虎旗下图片分享网站的用户朋友关系时变图数据集, 其中, 每个顶点表示一个用户, 而边 <(m, n), t>则表示用户 m 在时间 t 添加用户 n 为朋友. 该数据集包含了时间跨度为一个月的 Flickr 用户朋友关系网的演化情况, 即 31 个快照.

Weibo 是中国社交网站的社交追随者时变图数据集, 其含义与 Flickr 时变图类似. 该数据集包含了 27 天的社交网络关系网的演化情况, 即 27 个快照.

两个时变图数据集的更详细信息如表 3 所示.

表 3 测试数据集信息

数据集名称	总顶点数	总边数	快照数量	时间跨度 (天)
Flickr[16]	2 302 925	33 140 018	31	1
Weibo	1 787 443	423 071 214	27	1

2.4 实验设计与结果分析

2.4.1 延迟发送策略的执行效率

为分析延迟发送策略的执行效率, 我们分别在两个数据集上执行传统的标签传播算法和基于 NUMA 感知的延迟发送方法. 其中, 传统的标签传播算法采取了 NUMA 感知的时变图数据内存布局. 图 4 展示了两种方法求解连通分量的计算时间.

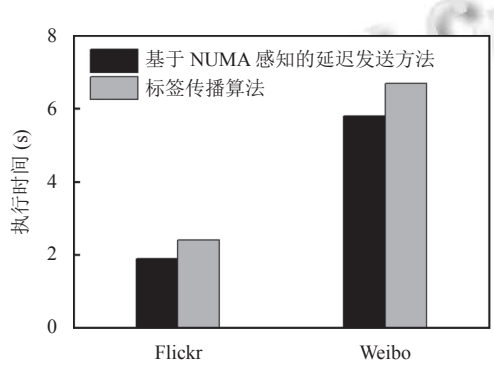


图 4 两种方法求解连通分量的计算时间

如图 4 所示, 基于 NUMA 感知的延迟发送方法在两个数据集的计算效率均比标签传播算法高. 其中, 在 Flickr 数据集上, 我们方法的计算时间约为 1.90 s, 而标签传播算法的计算时间约为 2.38 s, 我们方法的性能比标签传播算法提升 20.1%, 而在 Weibo 数据集上性能提升 15.23%. 主要原因是: 当执行标签传播算法时, 顶点能够直接访问跨 NUMA 节点的邻居, 产生大量远程随机访问, 而较高的远程访问延迟导致计算效率低. 与标签传播算法不同, 我们的方法首先在 NUMA 节点内部计算至局部收敛, 然后进行 NUMA 节点间的全局数据交换, 有效减少远程内存访问的数量, 从而大幅提升计算速度.

2.4.2 基于 NUMA 感知的延迟发送方法的效率

为了评估基于 NUMA 感知的延迟发送方法的效率, 我们将该方法与两个流行的图处理系统 Polymer 和 Ligra 进行对比实验. 由于 Polymer 和 Ligra 并不支持将整个时变图的多快照同时加载到内存进行计算, 我们将实验性能指标量化为平均每快照的计算时间.

如图 5 所示, 我们的方法在两个数据集的平均每快照计算时间均低于 Ligra 和 Polymer. 在 Flickr 数据集, Ligra 的平均快照计算时间约为我们方法的 2.5 倍, Polymer 约为我们的 1.3 倍, 而在 Weibo 数据集, Ligra 和 Polymer 分别是我们方法的 2 倍和 1.6 倍. Polymer 和我们方法的平均快照计算时间均少于 Ligra, 主要原因是两者都针对 NUMA 体系结构进行优化. 实验结果表明, 我们方法在性能比 Ligra 和 Polymer 具有明显的优势.

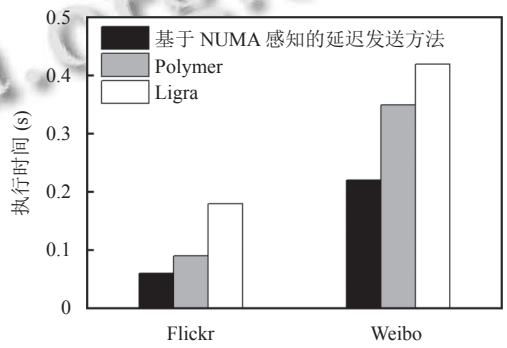


图 5 3 种方法的平均快照计算时间

2.4.3 NUMA 感知的时变图数据内存布局的有效性

我们在 Fickr 数据集上评估我们的方法、Ligra 以及 Polymer 的内存开销, 并使用 Linux 操作系统的性能分析工具 perf 在 Weibo 数据集上检测三者的 cache 未命中数量. 图 6 为 3 种方法在 Flickr 数据集的平均快照内存开销以及在 Weibo 数据集的 cache 未命中数量.

3 种方法在 Flickr 数据集的平均快照内存开销如图 6(a) 所示. 我们方法的平均快照内存开销为 142 MB, 而 Polymer 和 Ligra 的平均快照内存开销分别为 1180 MB 和 1360 MB. 尽管 Ligra 和 Polymer 采用 CSR[17] 对单张快照进行了压缩, 但是仍产生高昂的内存开销. 而我们的方法针对快照间的高度相似性, 对所有快照的边集合作并集运算, 避免了不同快照间边的重复存储, 有效节省内存开销.

此外, 图 6(b) 为 3 种方法在 Weibo 数据集的 cache 未命中数量. 在求解 27 张快照的 Weibo 数据集中, 我们的方法 cache 未命中情况为 2.6×10^9 次, 而 Polymer 和 Ligra 的 cache 未命中情况分别为 3.8×10^9 次和 5.1×10^9 次, 是我们方法的 1.46 倍和 1.96 倍. 正如第 1.2.2 节分析的, 由于顶点标签数据采取时间局部性设计, 顶点访问所有快照的邻居只会造成少量的 cache 未命中, 有效提高计算速度. 因此, 尽管我们的方法和 Polymer 都针对 NUMA 体系结构进行优化, 但是大量的 cache 未命中导致 Polymer 的性能比我们的方法低.

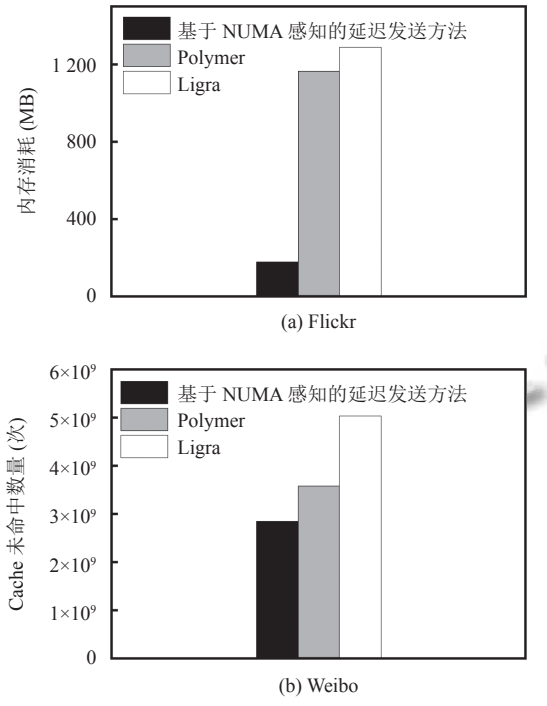


图 6 平均快照内存开销和 cache 未命中数量

2.4.4 额外内存开销

我们同样对缓存顶点标签副本的内存消耗进行了评估. 正如第 1.2.1 节讨论的, 为了在计算过程中减少远程访问的产生, 各个 NUMA 节点都会存储其他 NUMA 节点的顶点标签副本, 顶点能够直接访问本地副本进行数据更新. 我们分别比较了两个数据集的内存总消耗和标签副本的内存总消耗.

如图 7 所示, 在 Flickr 数据集, 图计算任务所需的总内存消耗为 4.40 GB, 但是标签副本的内存总消耗为 0.544 GB, 约占总内存消耗的 12.26%. 而在 Weibo 数据集, 图计算任务所需的总内存消耗为 36.4 GB, 标签副本的内存总消耗为 0.368 GB, 约占总内存消耗的 1%. 实验结果表明, 标签副本的内存消耗远小于方法执行过程的内存总消耗. 通过实验, 我们还观察到: 在 Flickr 数据集上执行图算法时, 标签副本的内存开销比例更高. 主要原因如下: 在 Weibo 数据集上, 每个顶点在每张快照的平均出度远大于 Flickr 数据集上的每个顶点的平均出度. 在 Flickr 数据集, 每个顶点在每张快照的平均出度约为 14, 而在 Weibo 数据集, 每个顶点在每张快照的平均出度约为 236. 巨大的平均出度导致巨大的总内存消耗, 但是标签副本的内存消耗只与顶点数量、NUMA 节点数量有关.

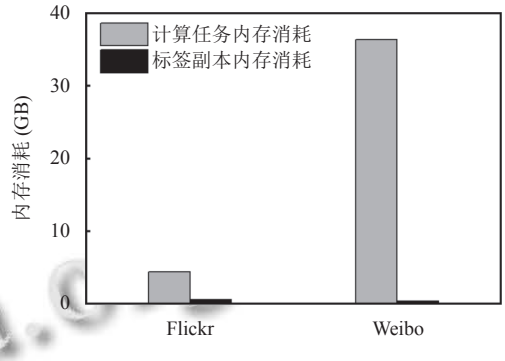


图 7 两个数据集的内存消耗情况

3 结语与展望

时变图的连通分量已经被广泛应用在不同的现实场景, 然而多数图计算方法都忽视了 NUMA 体系架构对图计算效率的影响, 造成大量远程随机访问, 导致执行效率低. 本文针对时变图的弱连通分量求解问题, 提出一种基于 NUMA 延迟发送的时变图弱连通分理求解方法, 它通过合理的数据内存布局, 合理控制 NUMA 节点间的信息交换次数, 最大限度减少远程内存访问数量, 显著提高了算法执行效率. 然而, 在全局同步阶段两次扫描的并行性依赖快照数量, 在一定程度上受到快照数量的限制. 因此, 提高全局同步阶段方法的并

行性将是下一步工作的重点.

参考文献

1 Kumar P, Huang HH. GraphOne: A data store for real-time analytics on evolving graphs. ACM Transactions on Storage, 2019, 15(4): 29.

2 吴安彪, 袁野, 乔百友, 等. 大规模时序图影响力最大化的算法研究. 计算机学报, 2019, 42(12): 2647–2664. [doi: 10.11897/SP.J.1016.2019.02647]

3 赵萍, 寿黎但, 陈珂, 等. 面向局域检索的时变图数据存储与查询模型. 计算机科学, 2019, 46(10): 186–194. [doi: 10.11896/jsjkx.19100530C]

4 王一舒, 袁野, 刘萌, 等. 大规模时序图数据的查询处理与挖掘技术综述. 计算机研究与发展, 2018, 55(9): 1889–1902. [doi: 10.7544/issn1000-1239.2018.20180132]

5 Kashef R, Pun H. Predicting *l*-CrossSold products using connected components: A clustering-based recommendation system. Electronic Commerce Research and Applications, 2022, 53: 101148. [doi: 10.1016/j.elerap.2022.101148]

6 李捷, 陈阳, 刘红霞. 基于社交效用向量的机会网络路由算法. 河南大学学报 (自然科学版), 2016, 46(2): 196–201. [doi: 10.15991/j.cnki.411100.2016.02.009]

7 杨懿轩. 基于实际路网的连通可靠性及关键路段判别与分析 [硕士学位论文]. 成都: 西南交通大学, 2019.

8 Feng GY, Ma ZX, Li DX, *et al*. RisGraph: A real-time streaming system for evolving graphs to support sub-millisecond per-update analysis at millions ops/s. Proceedings of the 2021 International Conference on Management of Data. ACM, 2021. 513–527. [doi: 10.1145/3448016.3457263]

9 Tan JS, Wang FZ. Optimizing virtual machines scheduling on high performance network NUMA systems. Proceedings of the 2017 3rd IEEE International Conference on Computer and Communications (ICCC). Chengdu: IEEE, 2017. 821–825.

10 Yang Z, Zhang AQ, Mo ZY. JArena: Partitioned shared memory for NUMA-awareness in multi-threaded scientific applications. arXiv:1902.07590, 2019.

11 Shun JL, Blelloch GE. Ligra: A lightweight graph processing framework for shared memory. Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. Shenzhen: ACM, 2013. 135–146.

12 Zhang KY, Chen R, Chen HB. NUMA-aware graph-structured analytics. Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. San Francisco: ACM, 2015. 183–193.

13 Vora K, Gupta R, Xu GQ. Synergistic analysis of evolving graphs. ACM Transactions on Architecture and Code Optimization, 2016, 13(4): 32.

14 Han WT, Miao YS, Li KW, *et al*. Chronos: A graph engine for temporal graph analysis. Proceedings of the 9th European Conference on Computer Systems. Amsterdam: ACM, 2014. 1.

15 Tarjan RE. A class of algorithms which require nonlinear time to maintain disjoint sets. Journal of Computer and System Sciences, 1979, 18(2): 110–127. [doi: 10.1016/0022-0000(79)90042-4]

16 Mislove A, Koppula HS, Gummadi KP, *et al*. Growth of the Flickr social network. Proceedings of the 1st Workshop on Online Social Networks. Seattle: ACM, 2008. 25–30.

17 Rodrigues J. GraphChi: Large-scale graph computation on just a PC. Computing Reviews, 2013, 54(8): 498.

(校对责编: 牛欣悦)