计 算 机 研 究 与 发 展
Journal of Computer Research and Development

DOI: 10.7544/issn1000-1239.202110839
60(3): 654−675, 2023

分布式多维大图迭代计算性能优化方法

杜玉洁[1] 王志刚[1] 王 宁[1,2] 刘芯亦[1] 衣军成[3] 聂 婕[1] 魏志强[1] 谷 峪[4] 于 戈[4]

[1](中国海洋大学计算机科学与技术学院 山东青岛 266100)

[2](密码技术与信息安全教育部重点实验室(山东大学) 山东青岛 266237)

[3](青岛市大数据中心 山东青岛 266071)

[4](东北大学计算机科学与工程学院 沈阳 110819)

(duyujie1201@163.com)

Optimization Methods for Distributed Iterative Computing Performance over Multi-Dimensional Large Graph

Du Yujie[1], Wang Zhigang[1], Wang Ning[1,2], Liu Xinyi[1], Yi Juncheng[3], Nie Jie[1], Wei Zhiqiang[1], Gu Yu[4], and Yu Ge[4]

[1](*School of Computer Science and Technology*, *Ocean University of China*, *Qingdao*, *Shandong* 266100)

[2](*Key Lab of Cryptologic Technology and Information Security*(*Shandong University*), *Ministry of Education*, *Qingdao*, *Shandong* 266237)

[3](*Qingdao Big Data Center*, *Qingdao*, *Shandong* 266071)

[4](*School of Computer Science and Engineering*, *Northeastern University*, *Shenyang* 110819)

Abstract Complex mining algorithms over large-scale graphs usually require highly frequent iterative analysis, while distributed computing with scalable computing power and storage capacity is a preferred solution for efficiency. However, the edges freely connecting vertices generate a lot of messages across distributed computing tasks and hence incurs communication costs during iterations, which heavily limits the performance improvement of distributed computing. To alleviate the negative impact of communication bottleneck, existing research basically employs combining and replicating techniques on top of the traditional pushing framework design. However, they mainly focus on easy-to-be-optimized single-dimensional-message algorithms with simple message data structure, and are not suitable for other important multi-dimensional-message algorithms with complex structure. Also, they cannot be seamlessly integrated into the-state-of-the-art pulling framework where messages are generated on demand. We thereby propose a lightweight vertex replication mechanism to synchronize replicated vertices and generate messages based on such replications on demand. The mechanism can work well under the pulling framework with inherent advantages in terms of fault-tolerance and memory consumption, and also greatly optimize the communication costs. Moreover, by considering the communication benefits and the costs incurred by possible workload imbalance, it can select an optimal replication threshold for best performance. Finally, extensive experiments over various real-world graphs validate the effectiveness of the lightweight vertex replication framework and the threshold analysis model.

收稿日期: 2021-08-20; **修回日期**: 2022-03-02

基金项目: 国家自然科学基金项目(61902366, 61902365, 62072083); 中央高校基本科研业务费专项资金(202042008); 山东大学密码技术与信息安全教育部重点实验室开放课题; 中国博士后科学基金项目(2020T130623); 青岛市自主创新重大专项(20-3-2-12-xx); 中国海洋大学计算机系研究生专业发展基金项目(CSZS2021003)

This work was supported by the National Natural Science Foundation of China (61902366, 61902365, 62072083), the Fundamental Research Funds for the Central Universities (202042008), the Open Project from the Key Lab of Cryptologic Technology and Information Security (Shandong University), Ministry of Education, the China Postdoctoral Science Foundation (2020T130623), the Qingdao Independent Innovation Major Project (20-3-2-12-xx), and the Graduate Professional Development Fund Project of Department of Computer Science of Ocean University of China (CSZS2021003).

通信作者: 王志刚(wangzhigang@ouc.edu.cn)

Key words distributed graph iterative computing; graph algorithms with multi-dimensional messages; communication optimization; vertex replication; workload imbalance

摘 要 大规模图的复杂挖掘算法通常需要高频迭代分析，而在计算与存储方面扩展性良好的分布式计算是提高处理效率的有效方案. 然而，图顶点之间存在自由分布的边关系，会在分布式计算任务之间产生大量消息，由此在迭代过程中产生的巨大通信开销严重制约性能收益. 已有工作在传统消息推送框架下采用合并和备份等技术降低通信代价，但主要面向结构简单、易优化的单维消息类算法，并不适用于结构复杂的多维消息类算法，也难以与当前最先进的消息按需拉取框架兼容. 因此提出一种新型轻量级顶点备份机制，通过备份顶点的按需同步以及本地消息的按需生成，可完美继承拉取框架在容错和内存管控等方面的系统优势，同时显著降低通信代价. 此外，通过考虑通信收益与负载偏斜代价，可计算最优阈值以提高整体性能. 最后在大量真实数据集上验证了相关技术的有效性.

关键词 分布式图迭代计算；多维消息图算法；通信优化；顶点备份；负载不均衡

中图法分类号 TP391

作为计算机科学中一种重要的数据结构，图可以表示现实世界中各种元素间复杂的关系，例如互联网中的社交网络、生物学中的蛋白质网络等. 随着大数据时代的到来，图数据的规模呈爆炸式增长，截至 2021 年 1 月，Facebook 的月活跃用户已超过 28 亿[1]，而用户之间复杂的社交关系导致边的规模更为庞大，这需要分布式处理框架提供可扩展的存储和计算能力[2]. 然而，图数据的各种应用分析通常需要进行高频迭代以逐步逼近最优解，而迭代过程中需要以消息的形式交换顶点之间的中间计算结果. 由于顶点可能分布于不同的分布式任务，这会产生大量的通信开销.

常见的图应用分析既包括网页排名计算 PageRank 和单源最短路径（single-source shortest path, SSSP）等简单算法，又包括社团分类 SC（semi-cluster）[2] 和复杂的多源最短路径计算（multi-source shortest path, MSSP）[3] 等复杂算法. 其一条消息的结构均包括目的顶点标识符以及消息值，但简单算法的消息值仅需一个基本数据类型即可表达，即单维消息值，如以浮点型数据表示 PageRank 的网页排名分数或 SSSP 的最短距离值；复杂算法则需要若干基本数据类型联合表达，即多维消息值，如以浮点型数组来表示 MSSP 中若干源顶点的最短距离值，以整型数集合表示 SC 中一个聚类内包含的实体等. 面对海量图数据的迭代处理作业，多维算法显然会急剧增加消息通信开销，严重制约分布式计算的性能收益.

为提高计算和存储的可扩展性，大量分布式图计算平台已经被开发出来并从通用性、易用性、健壮性和性能提升等各个方面进行了优化、完善. 其中关于通信优化的技术主要包括图划分[2,4-6] 以及给定划分后的消息合并[2] 与顶点备份机制[7]. 图划分作为一个 NP 完全问题[8]，难以在降低通信开销的解耦合和环节水桶效应的负载均衡方面实现综合最优. 因此，如何在给定划分结果的前提下进行通信优化，显得格外重要.

现有分布式图计算系统中的消息管理框架主要分为早期 Pregel[2] 与 GPS[7] 等系统采用的主动推送机制（push）和 PowerGraph[9] 以及 HGraph[10] 等系统采用的新型按需拉取机制（pull）. 已有的消息合并和顶点备份以及融合改进技术[11] 均在 push 框架下完成. 然而由于消息的目的顶点分布的局部性差，push 框架从机制上无法保证应合并消息被完全合并，严重制约实际性能收益；反之，pull 框架极大改善了局部性，能够保证应合并消息被完全合并，可最大化消息合并收益. 本文分析发现，对于 PageRank 类算法，pull 框架下的消息合并与顶点备份，在理论上可产生相同的性能收益. 然而，对于多维消息算法，如 MSSP，即使对某个源顶点相关的单维度消息进行了完全合并，不同源顶点所构成的多维消息值依然较大；而对于 SC 等算法，受计算逻辑正确性约束，仅能合并消息的目的顶点标识符而不可合并消息值. 因此，需在 pull 框架下实现顶点备份机制，在保留非备份顶点消息合并（或仅合并目的顶点标识符）收益的前提下，通过顶点备份进一步优化通信性能.

然而，现有顶点备份方法均在 push 框架下开发完成，其备份顶点值的同步策略依然采用 push 方式，如果直接迁移到 pull 框架下，会导致同一个迭代步中同时存在 push 与 pull 这 2 种消息管理体制，破坏原有 pull 框架的系统完整性与优化设计，比如高效的容错管理以及较低的内存资源消耗等特性. 此外，

备份机制虽然可带来通信收益,但会导致边数据在不同分布式任务之间进行迁移,影响原图划分结果的负载均衡,加重分布式环境下水桶效应导致的延迟开销.因此如何选择一个较好的备份控制阈值,对于获取最优的综合性能至关重要.此外,对于 MSSP 类支持合并的算法,迁移边会破坏消息合并所依赖的图结构,降低合并收益,如何在合并收益与备份收益之间进行均衡,是另外一个巨大挑战.

围绕多维消息算法的通信优化问题,本文针对平衡合并收益与备份收益的挑战,在新型 pull 框架[10]下设计了轻量级顶点备份机制,采用按需同步备份顶点值和优先级拉取消息等策略,使顶点备份与 pull 框架完美兼容;设计代价收益模型以均衡通信收益与偏斜延迟的影响,可根据数据集相关的线下先验知识和应用算法相关的线上实时信息,自动计算最优备份阈值,强化备份机制的实际性能收益并避免繁琐的手工阈值测试与选择操作;针对 MSSP 类可合并多维算法,从合并收益与备份收益 2 个角度分析多源点并发数目的取值,以确保备份机制的性能收益.大量真实数据集上的实验结果表明,传统 push 备份机制的内存开销均大于较本文提出的轻量级备份框架,最高可达 15 倍;对比现有非备份的 pull 框架,本文框架可实现高达 53% 的性能收益;而代价分析模型则可有效选择较优的备份阈值,实现与手动调整相近的性能收益.

1 相关工作

通信开销一直是制约分布式图处理性能提升的关键因素.本节总结了当前主要的相关工作并阐述本文技术与它们的区别.

1)图划分优化.高质量的图划分算法旨在解耦图数据以减少子图间的关联关系,进而减少通信开销,同时确保各任务的负载均衡以减少并行计算的水桶效应[12].然而,图划分问题属于 NP 完全问题[8].简单的 Hash[2] 和 Range[10] 划分可分别保证顶点和边数据的均衡分配,虽然顶点或边的切割会引起较高的通信开销,但由于划分速度快,已成为当前主流的划分机制.此外,多级层次划分算法如 Metis[6], PaToH[13], KaHIP[14] 等通过反复迭代调整数据分配位置,可显著降低通信开销,但执行效率过低.而流式划分[5] 尝试均衡通信优化质量和划分执行效率.本文的通信优化机制是针对给定划分后的子图进行 2 次优化,因此兼容上述图划分技术.

2)消息传递优化.除图划分外,在迭代计算过程中也存在很多通信优化技术.谷歌的 Pregel 系统[2] 首先针对多对 1 结构提出消息合并策略.Pregel 的开源实现 GPS 系统[7] 则提出 LALP 策略,对高出度顶点进行边迁移并备份源顶点;而以此为基础,进一步的工作探讨了如何在备份迁移过程中保证负载均衡[14-16].Pregel+[11] 在消息合并基础上融合 LALP,并增加边迁移(即源顶点备份)阈值的讨论,以在合并与备份之间进行均衡.然而,上述系统均采用 push 机制,这是由于目的顶点分布的局部性差,消息合并收益较低.不同于 push 框架, PowerGraph[9], CGraph[17], HDRF[18] 等框架在数据加载过程采用顶点分割策略,并设计了对应的 GAS(gather-apply-scatter)迭代计算框架,可同时支持图算法和机器学习算法,其中 Gather 即 pull 机制的核心部件.然而,顶点切分引入大量内存开销[19],且 GAS 计算频繁触发顶点之间的同步操作,开销较大.为此, PowerLyra[20], GrapH[21], L-PowerGraph[22], LightGraph[23] 等分别从顶点切分策略与顶点间消息传递方面进行优化.最近提出的 HGraph 系统[10] 则给出了以块为单位进行消息拉取的新型 pull 框架,显著改善了消息目的顶点分布的局部性,可实现完全彻底的消息合并,在不进行顶点切分的前提下,针对值合并类算法,其性能显著优于传统 pull 框架且在内存消耗与容错控制方面均有较大优势.然而,多维算法由于其消息值本身的字节规模较大,使得 HGraph 在彻底合并消息(值或目的顶点 *ID*)后,通信代价仍然较高,亟需通过顶点备份进一步降低相关开销.

近年来,基于特定硬件架构的图计算优化问题已经成为另一个研究热点[19,24],但本文关注通用架构下的通信优化,不对硬件条件进行特定假设.

2 问题定义

本节首先阐述分布式图迭代计算的一般处理方式;然后根据消息数据的维度以及合并属性对图迭代算法进行分类,并着重介绍近些年提出的具有重要实用价值的多维消息类算法;最后基于推送(push)和拉取(pull)这 2 种主流分布式消息管理框架,分析合并与备份对不同类型图算法的优化效果.

2.1 分布式图迭代计算

给定输入的有向图 $G=\langle V, E\rangle$,其中 V 为 $|V|$ 个顶点的集合而 E 为 $|E|$ 条边的集合.E 中每条有向边 $e=\langle v_i, v_j\rangle$ 连接源顶点 v_i 和目的顶点 v_j,其中 v_i 是 v_j 的入

度邻居/顶点，而 v_j 是 v_i 的出度邻居/顶点. 图以邻接表形式存储，每条邻接表包含顶点 v_i 和所有以 v_i 为源顶点的出边的目的顶点.

分布式图迭代计算系统在启动迭代计算之前，首先将 G 从初始存储位置（如分布式文件存储系统HDFS）并行加载到 P 个不同的分布式计算任务 T_i 上，每个任务负责处理一部分数据，即子图 $G_i=\langle V_i, E_i\rangle$，该过程即图划分；随后各任务 T_i 对本地子图 G_i 中的图数据进行迭代计算，计算过程中消息的生成、发送和接收处理都是按照出边进行的，相应顶点循环执行更新操作，每次循环即一个迭代步，迭代步之间通过全局同步路障来协调各个任务的处理进度. 第 k 个迭代步的具体操作包括：根据第 k−1 步接收的消息更新顶点值，将更新后的顶点值以消息的形式沿着出边发送给目的顶点，以便在第 k+1 步执行更新操作. 如果顶点 v_i 参加第 k 步迭代的更新计算，则称 v_i 在该迭代步是激活的，编程人员可根据需要设置激活标记，以避免非激活顶点的无效计算. 当所有顶点均处于非激活状态且系统中没有新的消息生成时，算法收敛，迭代计算结束.

2.2 多维消息类图迭代算法

依据分布式图算法在迭代计算过程中传递消息的不同维度特征和合并属性，可对分布式图算法进行分类. 具体分类标准包括：1）一条消息数据的消息值可以由单个基本数据类型独自表达或多个基本数据类型联合表达，即单维与多维；2）发往同一个目的顶点的不同消息值是否允许被合并为一个值，即值合并和连接. 表 1 展示了常见分布式图算法的分类结果. 下面以 SSSP、标签广播算法（label propagation algorithm, LPA）、MSSP 和 SC 为例，分别按照 2 种分类标准对不同类型算法的特征进行阐述.

Table 1 Graph Algorithm Classification

表 1 图算法分类

分类依据	单维	多维
值合并	PageRank CC（connected components） SSSP BFS（breath-first-search） PPR（personalized PageRank）	MSSP MBFS（multi-source BFS） MPPR（multi-source PPR）
值连接	LPA Louvain[25]	SC SA[26]（simulated advertisements） PSCAN[27]

SSSP 算法的目标是发现给定源顶点到图中其他所有顶点之间的最短距离. 迭代初始阶段，源顶点将顶点值（即距离值）初始化为 0 并根据出边的距离权重生成消息值发送给出度顶点，而其余所有顶点均将顶点值设置为无穷大. 随后，每步迭代过程中，收到上一步消息的顶点被激活并从入度邻居的消息值和自身顶点值中选择最小的值进行值更新，而如果顶点值发生了更新，则沿出边生成新消息并发送给出度邻居. 每条消息 $msg=\langle ID, msgValue\rangle$，其结构仅包括一个 int 型的目的顶点 *ID* 和 double 型距离值 *msgValue*，属单维消息. 此外，算法逻辑仅关心最短距离值，所以如果沿 2 条或多条目的顶点相同的出边如 $e_{13}=\langle v_1, v_3\rangle$ 和 $e_{23}=\langle v_2, v_3\rangle$，分别生成具有不同消息值的消息如 $msg_{13}=\langle 3, 0.1\rangle$ 和 $msg_{23}=\langle 3, 0.5\rangle$，则可合并为一条消息 $msg=\langle 3, \min\{0.1, 0.5\}=0.1\rangle$ 以节省通信开销.

LPA 是一种快速社团发现算法，其将每个顶点赋值一个社团标签并初始化为顶点 *ID*，随后迭代更新标签值为其入度邻居标签值中出现次数最多者. 由于顶点更新依赖所有入度邻居的标签值分布，所以每步迭代中每个顶点均参与更新且沿出边向所有出度邻居广播自己的标签值，即全激活. 与 SSSP 相比，其消息结构相同，即目的顶点 *ID* 和 int 型的标签值，属于单维消息；但不同之处是，由于需要根据标签频数分布进行更新，所有消息值不可合并，仅可连接，即如有 $msg_{13}=\langle 3, 2\rangle$ 和 $msg_{23}=\langle 3, 2\rangle$，仅可连接消息值为 $msg=\langle 3, [2, 2]\rangle$ 以合并（共享）目的顶点 *ID* 进而节省通信开销.

MSSP 是 SSSP 的一种常见多源点扩展. 高级图挖掘与分析算法通常需要衡量图中所有顶点对之间的最短距离，而通过串行提交不同源顶点的 SSSP 作业，会造成图的反复遍历，效率低下. 一种高效的解决方案是根据集群的计算与存储能力，在一个图遍历作业内并发计算多个源顶点的最短距离分布，即 MSSP. 假设并发源顶点个数为 m，则此时每个顶点值由一个 double 型数据扩展为长度为 m 的 double 数组；

对应地, 消息值也扩展为 double 数组. 例如, 当 m=3 时, 可有 msg_{13}=〈3, [0.1, 0.4, 0.2]〉和 msg_{23}=〈3, [0.5, 0.3, 0.1]〉, 此时虽然对应源顶点的消息值可合并, 但合并后的消息值仍是一个长度为 3 的数组, 即 msg=〈3, [0.1, 0.3, 0.1]〉, 故属于可合并、多维消息类算法. 其他单源点遍历算法如 PPR 和 BFS 均有类似的多源扩展.

SC 是谷歌开源图处理系统 Pregel 中内置的一种半聚类算法, 即允许一个顶点记录自己所属的多个聚类并打分排序. 迭代初始, 每个顶点将自身初始化为一个聚类并发送给出度邻居. 在每个迭代步, 所有顶点均激活, 根据入度邻居所属的聚类分布更新自己所属聚类, 并继续广播. 算法传播的消息是描述顶点所在的聚类(即顶点集合), 需要多个基本数据类型进行联合表达, 属于多维消息结构; 且由于需要聚类分布信息, 故消息值不可合并. 延续上例, 可有消息 msg_{13}=〈3, (1)|0.6,(2,5)|0.3〉和 msg_{23}=〈3, (2,5)|0.3〉, 即顶点 v_1 可归属于包含顶点 v_1 的聚类(1)和包含顶点 v_2 与 v_5 的聚类(2,5), 分数分别为 0.6 和 0.3, 而 v_2 则以 0.3 的分数归属于聚类(2,5). 与 LPA 类似, 2 条消息因对应的目的顶点相同故可以合并发送, 但消息值仅可连接, 即以 msg=〈3, [(1)|0.6, (2,5)|0.3,(2,5)|0.3]〉的形式进行发送.

综上, 对于单维值可合并类算法, 如果可以保证应合并的消息被全部合并, 可极大缓解通信压力; 对单维值连接类算法, 由于消息值不可合并, 每个迭代步中总的消息值规模最多可达到出边的数量级, 即 $|E|$, 但由于每个消息值的字节数较少, 故通信压力仍可以接受; 反之, 对于多维消息类算法, 其单条消息值的规模取决于联合表达所用的基本数据类型的数目, 即维度, 如 MSSP 算法中的并发源顶点数目和 SC 算法中描述聚类特征的基本数据类型集合规模. 在相同的输入图规模下, 多维算法显然会产生较大的通信压力. 而当消息值不可合并时, 通信代价更会急剧增大. 而根据已有测试结果, 在分布式图算法处理过程中, 即使对于单维消息类算法, 任务间通信引入的时间开销约占总执行时间的 50% 以上[3]. 因此亟需针对多维消息图迭代算法设计通信优化技术, 以提升分布式计算效益.

2.3 消息合并与源顶点备份收益分析

分布式通信问题可以通过提升图划分质量加以改善, 即在保证负载均衡的前提下尽量减少子图之间的边耦合依赖程度. 然而, 图划分是一个传统的 NP 完全问题[8], 难以在合理时间内获得高质量划分结果. 因此, 对已有划分后的子图进行后续通信优化就显得尤为重要. 目前的优化方法主要分为 2 类: 消息合并和顶点备份. 下面结合 push 和 pull 这 2 种消息传输方式分析 2 种优化方法产生的效益, 以突出本文研究的必要性.

2.3.1 push 与 pull 消息传输方式对比

迭代过程中消息的生成与传送方式可分为两大类设计, 分别为 push 和 pull. 在迭代步 k, push 在顶点更新时直接遍历所有出边并主动推送消息给所有目的顶点; 而 pull 仅完成顶点更新、不发送消息. 在迭代步 k+1, push 可确保目的顶点所需消息均已接收并储备在本地, 可直接使用; 而 pull 需要根据边的依赖关系从对应源顶点处按需拉取消息数据. 2 种消息管理框架各有优缺点, push 在一个迭代步中, 仅需遍历一次顶点即可完成顶点值更新和新消息生成, 但由于顶点之间边关系的自由分布, 一个顶点的出边所指向的目的顶点的分布具有较差的局部性, 即主动推送的消息数据所指向的目的顶点的局部性差, 且该部分消息直到下一个迭代步才被使用, 因此需要在发送端和接收端设置大量消息管理缓存, 需较多的内存资源; pull 由于消息按需生成且消息均指向欲更新顶点值的目的顶点, 故局部性良好, 且接收的消息可直接被目的顶点处理, 无需缓存, 极大节省了内存资源, 但不同目的顶点的更新会导致其共享的源顶点被随机扫描读取多次.

2.3.2 消息合并

当某个任务上的多个源顶点均需要向同一个目的顶点发送消息时(多对 1 结构), 如果消息值可合并, 显然可以在消息发送之前进行合并(如 2.2 节的 SSSP 算法), 以减少通信开销. 然而, 在 push 方式下, 考虑到消息的目的顶点分布的局部性较差而发送端缓存又是有限的, 因此能够在缓存中参与合并的消息比例较少, 即无法保证彻底的消息合并, 导致通信收益降低, 甚至难以抵消合并所引入的管理开销. 如图 1(a)所示, 假设发送端缓存容量为 2 条消息, 可保证顶点 v_1 与 v_2 发往目的顶点 d_1 与 d_2 的消息被合并; 但当 v_2 继续往 d_3 发送消息时, 由于缓存已满, 需将发往 d_1 与 d_2 的消息清空; 最后 v_3 往 d_2 发送消息, 但因缓存清空, 该消息无法与 v_1 与 v_2 生成的消息合并, 即应合并消息无法保证被彻底、完全地合并. 相反地, 在 pull 方式下[5], 如图 1(b)所示, 目的顶点按照 2 为单位进行分块, 然后以块为单位拉取所需消息. 第 1 个块中, 消息均发往 d_1 与 d_2, 局部性优异, 在缓存为 2 的前提下, 可被完全合并; 之后第 2 个块启动拉取操作. 此外, 这种按块拉取的方式, 可保证同一个块

内的目的顶点仅扫描 1 次对应的源顶点，降低源顶点的随机读取次数. 需要注意的是：这里的消息合并是泛化概念，即对于 2.2 节中值合并类算法，可实现目的顶点 *ID* 与消息值合并；而对值连接类算法，仅可实现目的顶点 *ID* 的合并，其通信收益仍在，但效果减弱.

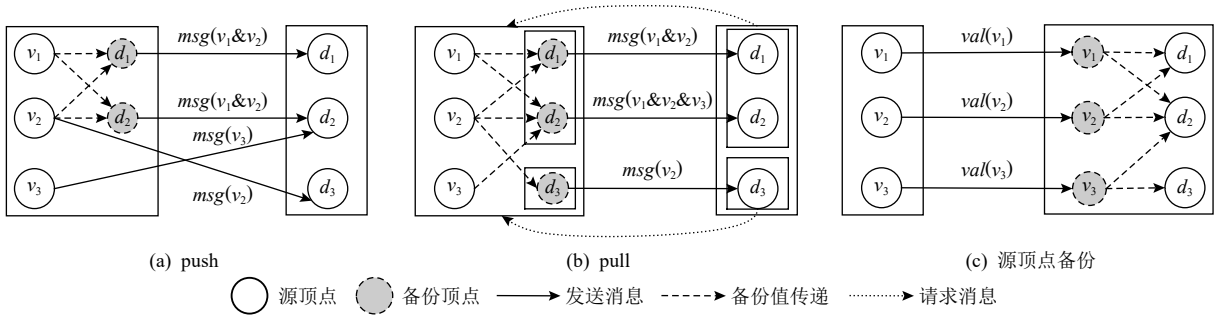


Fig. 1 Illustration of message combination and vertex replication

图 1 消息合并与顶点备份图示

2.3.3 顶点备份

当某个顶点的出度较高以至于其在若干任务上均有大量的目的顶点（1 对多结构），可以将出边迁移至目的顶点所在任务并对源顶点进行备份，从而将迁移边所对应的网络消息转换为目的顶点所在任务的本地消息，同时增加了同步备份顶点值的网络开销. 如图 1(c)所示，源顶点备份主要在传统的 push 框架下实现，如 GPS[7] 和 Pregel+[11]. 当顶点更新后，同步备份顶点的值，而备份顶点收到同步值之后立即沿迁移边生成本地消息. 同步值与本地消息均采用 push 方式管理. 考虑到迁移边的消息不再由源顶点所在任务生成，这会影响消息合并的机率. 因此，对于消息合并类算法，Pregel+[11] 设计了合并优先的备份机制，即只有当目的顶点的入度为 1 时，其对应的源顶点才可能被备份（此时无其他源顶点指向该目的顶点，故不会损失合并收益），以兼顾合并与备份的收益. Pregel+[11] 在非完全合并的 push 框架下取得了较好的通信压缩效果；但在新型 pull 框架下，由于彻底合并已经极大压缩了通信规模，根据我们的实测结果，如表 2 所示，虽然满足备份约束的顶点比例较高，但备份仅带来 1%~7% 的微弱压缩效果，而实际性能收益可以忽略. 表 2 所示的 4 个真实图数据集，包括互联网领域的 Web 图数据集 Uk2014tpd（UK）、Wikipedia（Wiki）和 Eu2015host（EU），以及社交网络领域的常用图数据集 LiveJournal（LiveJ）.

Table 2 Replicate Effect of Pregel+ Under Thorough Combination

表 2 Pregel+在彻底合并下的备份效果 %

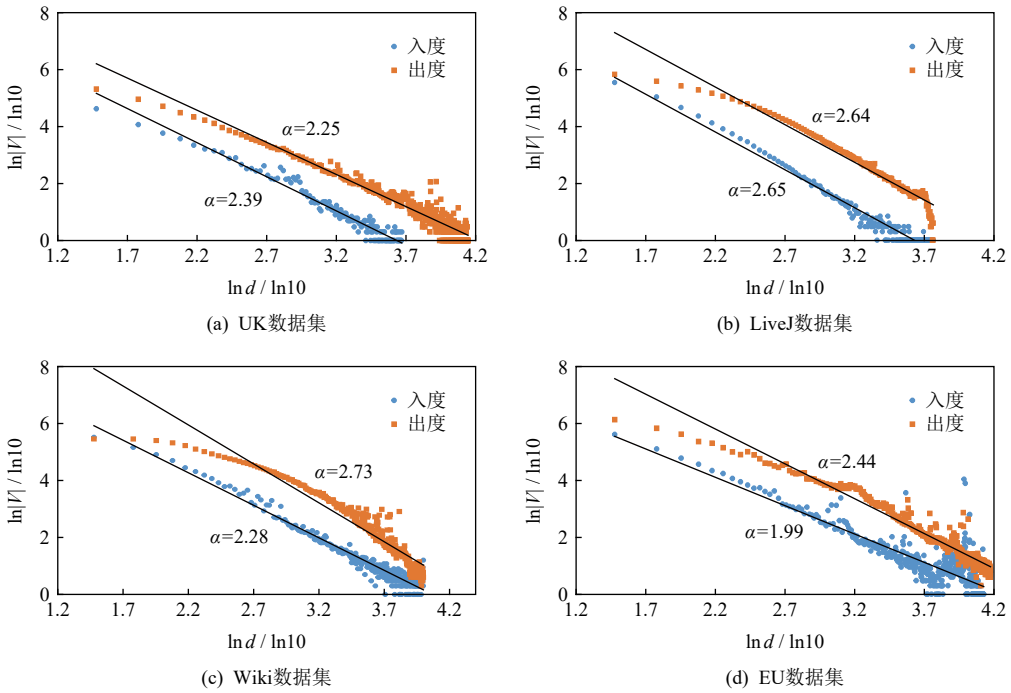
数据集	入度为 1 的目的顶点占比	通信优化百分比
UK	51.29	7.11
LiveJ	35.05	2.98
Wiki	31.81	1.47
EU	34.60	1.08

2.3.4 分析

对比消息合并和顶点备份机制可发现，对于合并类消息算法如 SSSP，在以块为中心的 pull 框架下，其完全彻底合并消息的特点特别适合多对 1 结构，合并后仅需发送一条消息. 本质上，可以看作目的顶点在源顶点所在任务上的备份过程（如图 1(b)所示），最终的网络通信代价取决于目的顶点的备份规模；另一方面，现有顶点备份适用于 1 对多结构，仅有备份顶点的同步会引入网络开销，通信代价取决于源顶点备份规模（如图 1(c)所示）. 因此，无论是对目的顶点还是对源顶点进行备份，备份后的通信规模都是由备份顶点的数量决定的. 注意到 PowerGraph[9] 提供了关于求解顶点 v 在 P 个任务上备份顶点数的期望公式：

$$E\left[\frac{1}{|V|}\sum_{v\in V}|A(v)|\right]=P-\frac{P}{h_{|V|}(\alpha)}\sum_{v=1}^{|V|-1}\left(1-\frac{1}{P}\right)^{d}d^{-\alpha},\qquad(1)$$

其中 d 为顶点 v 的出度或入度，V 是顶点集，$|V|$是顶点集中顶点个数，$|V|$与幂律偏斜指数α和 Zipf 分布的归一化常数$1/(h_{|V|}(\alpha))$直接相关. 其中，消息合并关心的目的顶点备份规模依赖入度偏斜，而源顶点备份规模则依赖出度偏斜. 图 2 分析了本文所用真实数据集的出入度偏斜指数，可以看出两者近似相等. 因此，不同于传统 push 框架下的非完全合并，对于值合并类算法，pull 框架下的消息完全合可带来与源顶点备份相近的通信收益；即使对于值连接类算法，pull 框架的优异合并效果依然可以在目的顶点合并方面带来性能收益.



(a) UK数据集 (b) LiveJ数据集

(c) Wiki数据集 (d) EU数据集

Fig. 2 The skewness of the in/out degree distribution for different datasets

图 2 各数据集的出/入度偏斜指数

特别地，对于 MSSP 类图遍历算法，顶点值是逐步收敛的. 在第 k 步迭代中，同一目的顶点所对应的多个源顶点中可能有顶点值已经达到收敛而停止更新，该部分顶点自然不会生成新消息. 这种算法逻辑层面的部分收敛现象显然会减少参与合并的消息规模，削弱多对 1 结构产生的合并收益. 相反地，顶点备份依赖 1 对多结构. 对于某个顶点而言，只要其尚未收敛，则需要继续沿出边广播消息，顶点备份可继续正常工作，不受顶点收敛的影响. 这会在两者之间产生通信收益差，且差值与消息值维度成正比，即 MSSP 类算法并发源顶点个数越多，2 种机制的通信收益差距越大. 基于上述分析以及 pull 方法极低的内存消耗特别适合大规模图数据处理，本文因此致力于在以块为中心的 pull 框架下，针对多维消息算法，通过源顶点备份机制进一步优化消息值的传输开销.

考虑到源顶点备份会破坏原有图划分的均衡负载分布，且现有源顶点备份机制均在 push 框架下设计，难以兼容新型 pull 框架，因此需要重新设计备份机制并仔细分析备份阈值，以实现功能性和性能优化方面的统一.

3 轻量级顶点备份框架

由于大部分算法的基本工作流程是顶点更新与边消息传递，而图中边的规模远大于顶点规模，故工作负载与边密切相关. 因此，本文假设采用简单快速的 Range 划分以保证划分后各任务间的出边数目均衡，然后通过对消息传递模型的改进降低网络通信量，以实现图处理性能的整体提升. 然而，已有消息传递模型的改进主要是基于非完全合并的 push 环境下进行的. 为实现通信开销的进一步压缩，本文在完全合并的 pull 环境下，即 HGraph[10] 系统上设计新的轻量级顶点备份机制，以改善多维消息算法的消息值传输效率.

轻量级顶点备份的核心是，在 pull 系统下，备份点的相关操作也采用 pull 方式实现；通过只使用一种 pull 管理方式，避免了传统 push 顶点备份机制在 pull 方式下内存开销大与容错负载重的问题，程序设计简洁、易维护. 本节首先总结 push 备份在 pull 框架下的缺点，然后介绍轻量级备份的按需同步和优先级消息拉取技术，最后对比本文备份框架与 PowerLyra 的混合备份技术，突出本文备份的轻量级特点.

3.1 push 顶点备份与 pull 消息管理框架的兼容性

根据 2.3 节中对消息合并与顶点备份的收益分析可知，在完全合并的 pull 框架下，两者对值合并类算法的通信压缩效果相近. 但针对多维算法，在保证完全合并(目的顶点 *ID* 合并、消息值合并)的前提下，可针对部分高出度顶点进行边迁移与源顶点备份，以进一步优化通信开销. 然而，目前源顶点备份对通信的改进都是在 push 框架下实现的，备份顶点的同

步以及迁移边的消息生成方式，均采用 push 主动推送方式，如果直接在 pull 框架下实现，会导致每个迭代步内同时存在非备份顶点的 pull 操作以及备份顶点的 push 操作，带来 2 个缺点：

1）容错管理复杂且效率低. 容错控制对图迭代计算至关重要，可在部分任务发生故障时避免其他任务回滚、重新计算. 目前的容错机制主要采用检查点回滚的方式进行故障恢复. 为避免非故障任务的重新计算，需要不断记录每个任务的消息输出，以便故障任务在重新计算时使用. 大量的消息记录，尤其是多维算法的大消息值特性，会严重影响正常迭代的计算效率. 在 push 方式下，由于消息是主动生成并发送出去，无法对此进行优化；而 pull 方式允许消息按需生成，故可按需生成故障任务恢复过程中所需的消息，不必主动记录. 当故障节点需要入度邻居的消息来更新顶点值时，仅需沿入边向入度邻居发送拉取请求，而非故障任务上的顶点只需记录对应迭代步的顶点值以响应消息请求即可. 由于顶点的规模远低于消息规模，记录顶点值对正常迭代计算的影响很小. 然而，一旦 pull 与 push 混合执行，则仍需要对 push 方式下的顶点同步值以及根据迁移边生成的消息进行记录，既增加了容错管理的复杂性，也增大了容错开销.

2）多缓存高内存消耗. 使用 push 方式进行消息发送时，需要在发送端针对每一个分布式任务设置一个双缓存结构，以便其中一个缓存溢出、进行消息发送时，另一个缓存可继续接收消息、不阻塞顶点的计算更新；在接收端，由于消息对应的目的顶点的局部性差且无法预知其到达时间，需要根据目的顶点的分块信息、消息源顶点所在的任务等设置多个缓存区，以避免针对同一个目的顶点的消息进行整理时导致频繁的加锁开销. 在多维消息类算法中，由于顶点值以及据此生成的消息值的规模巨大，发送端与接收端的多缓存设置给内存造成巨大压力. 而在 pull 系统中，消息按需生成且生成之后被立即消耗，因此根据需要更新的目的顶点的规模预估消息规模设置缓存即可，避免了繁杂的多缓存结构，节省了内存开销. 同理，当 pull 与 push 混合时，为正确、高效运行 push 机制，需要配备多个缓存结构，增大了内存开销. 因此，需要设计与 pull 机制相兼容的顶点备份框架，在实现轻量级程序框架设计的同时，可以实现容错和内存管理的简洁与高效.

3.2 基于按需同步的备份更新策略

鉴于 push 备份与 pull 框架的冲突点是由备份点的同步方式以及迁移边的消息生成方式所导致的，因此需要将这 2 种方式改为 pull 方式，以实现系统兼容. 本节重点介绍基于按需同步的拉取式顶点备份机制.

3.2.1 按需同步框架设计

执行顶点备份后，每个迭代步中顶点值计算更新所需的消息来源于 2 个部分：1）所有任务上非备份顶点发送的非备份消息值；2）备份到本地的顶点根据迁移出边发送的本地备份消息值. 当目的顶点块欲执行更新操作时，针对非备份消息值，直接以块为单位向所有任务发送拉取请求，而各任务接收请求后，直接扫描本任务内指向该请求块内所有目的顶点的出边，生成所需消息并在发送端执行彻底合并后发送给请求端（即目的顶点块所在的任务），该过程可由现有 pull 机制直接完成；而针对本地备份消息值，在按需同步策略下，某个顶点值被更新后，不会主动推送消息以同步其备份顶点值，因此应先同步其备份值，然后生成本地备份消息. 具体地，在同步备份顶点值时，仍然以目的顶点块为单位向所有任务广播同步请求，而各任务收到请求后，检索本地顶点是否有指向该请求块内目的顶点的迁移出边（即是否有备份），如是，则响应同步请求，将顶点值发送到请求端，然后根据同步后的备份顶点值生成本地备份消息. 进一步地，为实现按需生成本地备份消息的目标，需将备份的源顶点和迁移的出边以邻接表的形式分块存储，即每个迁移过来的邻接表按照目的顶点所在的块对迁移边进行分割存储. 如是，当目的顶点所在的块欲执行更新操作而拉取消息时，仅需读取每个迁移邻接表中对应该块的部分出边即可生成所需消息，从而避免 push 方式下的多种缓存设置，降低内存消耗.

3.2.2 2 阶段同步响应优化

在同步响应过程中，各任务的检索操作需要遍历所有出边，时间复杂度较高. 此外，某个源顶点的出边可能指向任务 T_i 上不同块内的目的顶点. 当任务 T_i 上不同块内的目的顶点发送同步请求时，该源顶点所在的任务需进行多次检索以及响应操作，造成备份顶点值的冗余同步，浪费计算和网络资源. 为提高同步效率，本文设计了基于字典的 2 阶段同步响应机制. 具体地，每个任务在内存中维护同步响应字典，即如果某个顶点在某个任务上存在备份，则在字典中添加该条记录，且标记该备份值在当前迭代步是否已经被同步. 根据 2 阶段同步响应机制，当某个目的顶点块欲向任务 T_i 发送同步请求时，其首先

查验本地是否存在来自于 T_i 的迁移边，如果没有，显然无备份顶点，无需发送请求；如存在，则正常发送请求. 任务 T_i 收到请求后，首先查验响应字典，如果指向请求端所在任务存在备份点且所有备份点均已被同步，则不再响应，返回值为空；否则，根据字典中尚未标记同步的顶点查找顶点值以响应同步备份顶点值并更新字典内容. 2 阶段同步机制显然可以根据字典信息避免冗余同步，提高响应效率.

3.2.3 实例演示

图 3 展示了按需同步策略下数据存储和管理方式的一个实例. 该实例包含 20 个顶点(图 3 中顶点编号直接以数字形式呈现)，分布于 2 个分布式任务 T_1 与 T_2. 以 T_1 为例，本地图数据包含顶点 v_1 至 v_{10} 及其邻接表，具体分为 2 个块，分别包含顶点 v_1 至 v_6 和 v_7 至 v_{10}. 对应地，出边按照目的顶点的分块信息按列分割存储. 如 v_1 的出边指向 4 个目的顶点，其中目的顶点 v_2 和 v_3 属于同一个顶点块，故对应出边被存储到同一列中；同理，v_7 和 v_{18} 分别属于不同顶点块，故对应出边被存储在另外 2 列中. 特别地，v_6, v_7, v_{10} 分别有边迁移到任务 T_2，即在 T_2 上存在备份. 以为 v_6 例，其出边〈v_6, v_{13}〉和〈v_6, v_{15}〉被迁移到 T_2 并按照目的顶点的分块信息进行按列分割存储，而迁移之后，T_1 的响应字典中应添加 1 条 v_6 指向 T_2 的记录. 在第 k 步迭代中，假设 T_2 上的顶点块 v_{11} 至 v_{15} 欲更新顶点值，则分 2 步拉取所需消息：1)本地备份消息，即首先检查本地是否有来自 T_1 的迁移边，如有，则向 T_2 发送同步请求，而 T_1 收到请求后，首先检查字典中 T_1 对应的列是否均为 1，否则，如有且字典中对应值为 0(如此处的 v_6 与 v_7)，则应封装对应顶点值进行响应并将字典中的值更新为 1(此处即 v_6 与 v_7 在 T_2 列的值)，而 T_2 收到同步值之后根据迁移边按需生成本地备份消息；2)非备份消息，可按照原有 pull 框架设计，发送消息拉取请求并通过扫描对应列的出边信息生成消息并返回给 T_2. 当顶点块 v_{16} 至 v_{20} 被调度更新时，可重复此过程，但需要注意的是，v_{16} 的更新依赖于 v_7 的消息，但 v_7 的顶点值已经被同步(即 T_1 中字典的对应值为 1)，故该值不会被再次返回，以避免冗余同步，减少网络通信开销.

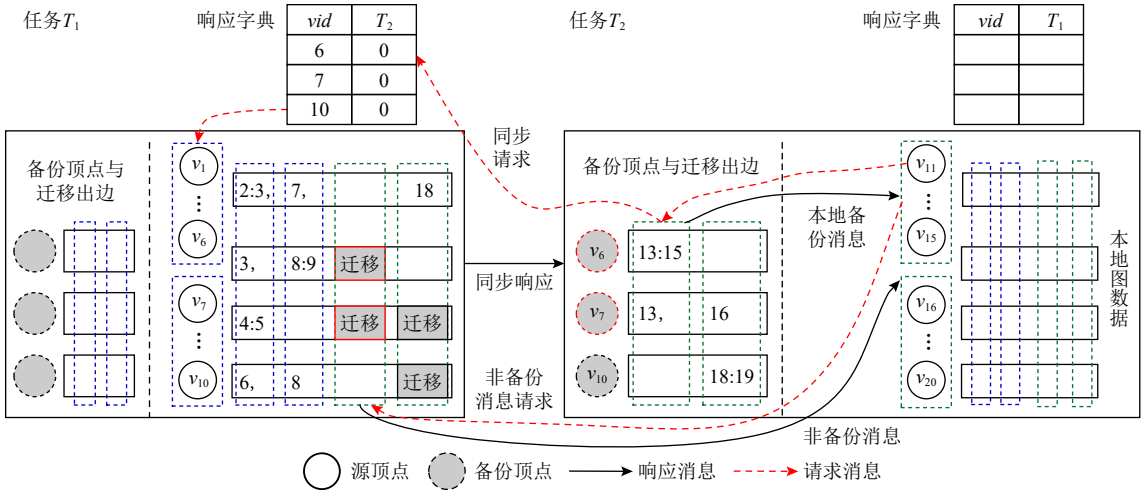


Fig. 3 The data storage and management methods of on-demand synchronization update strategy

图 3 按需同步更新策略的数据存储和管理方式

3.3 基于优先级拉取的并发消息生成

在按需同步备份更新策略下，顶点更新所依赖的消息包括备份消息和非备份消息. 为获取这 2 类消息，直观的解决方案是并发发送备份值同步请求和非备份消息请求(详见图 3 示例). 然而，这种方案的弊端是顶点同步值和非备份消息值的同时传输会增大瞬时通信负载，造成网络拥堵；而在目的任务接收到响应的备份顶点同步值和非备份消息值后，迭代计算的负载重心转为备份消息的本地生成以及目的顶点更新，均不涉及网络通信，导致网络资源空闲. 本节介绍基于优先级拉取的并发消息生成策略，通过备份顶点值和非备份消息值的错峰拉取，提高网络资源使用效率.

3.3.1 优先级错峰拉取

基于优先级错峰拉取和并发拉取的区别在于，前者优先拉取备份顶点的同步值，然后拉取非备份消息且同时启动本地备份消息的生成与合并处理工作，最后待所有需要的消息准备完毕后，进行目的顶点的计算更新. 该方案的优点是不同优先级的拉取请求错峰响应，消息在网络中的传输压力减小，且减

少了空闲等待状态，充分利用网络通信带宽.

3.3.2 优先级动态调整

给定一个目的顶点块，由于响应字典的存在以及算法本身消息规模的动态变化，会导致需要拉取的备份顶点同步值以及非备份消息值的规模动态变化. 直观地，当两者的规模较低时，优先级错峰拉取可能导致两者各自均无法充分利用通信带宽，降低网络资源使用效率. 式(2)和式(3)分别描述了并发拉取和优先级拉取的性能度量方法.

$$\varphi_{\text{con}} = \lambda \cdot \max\left\{\varphi_{msg}, \varphi_{\text{syn}} + \varphi_{\text{pro}}\right\}, \tag{2}$$

$$\varphi_{\text{pri}} = \varphi_{\text{syn}} + \max\left\{\varphi_{msg}, \varphi_{\text{pro}}\right\}. \tag{3}$$

无论采用何种拉取策略，具体工作负载均包括拉取非备份消息值的开销φ_{msg}和拉取本地备份消息的开销，而后者可细分为同步备份顶点值开销φ_{syn}和本地备份消息生成开销φ_{pro}. 对于并发拉取，由于2种拉取请求同时发送、同时响应，故其性能指标φ_{con}取决于φ_{msg}与$\varphi_{\text{syn}}+\varphi_{\text{pro}}$中的较大值，而考虑到同时请求产生的网络拥堵，应添加惩罚因子$\lambda(\lambda \geqslant 1)$; 对于优先级拉取，同步请求被优先发送，而后并行执行备份消息的拉取以及本地备份消息的生成，故其性φ_{pri}能应在φ_{syn}的基础上累加后两者的最大值，即$\max\left\{\varphi_{msg}, \varphi_{\text{pro}}\right\}$. 当$\lambda$较低，比如$\lambda=1$时，显然有$\varphi_{\text{con}} < \varphi_{\text{pri}}$，即并发拉取优于优先级拉取；反之，当$\lambda$较高，即备份顶点值和非备份消息值规模较大而导致网络拥塞程度加剧时，优先级拉取优于并发拉取. 在实际运行图迭代计算时，可根据算法的执行进度和网络瞬时状态，实时计算φ_{con}与φ_{pri}的对比结果进而选择决定是否将同步请求的优先级升高. 具体地，对于需要同步的备份顶点值的规模，可在请求发起端(如图3中的任务T_2)记录当前迭代步已经完成同步的备份顶点，当启动一个新的目的顶点块的更新时，可先分析本地迁移边以确定需要同步的备份顶点数目，然后对比已经完成同步的备份顶点，以估算同步开销；同理，可根据本地迁移边规模估算本地备份消息的生成开销；对于非备份消息的规模，因该类消息由其他所有任务生成，相关统计信息无法在本地获取，故可在迭代计算过程中，通过记录上一个迭代步中获取的消息规模来估算本步的消息规模[28]；而对于λ，可通过测试给定集群在不同拥塞程度下的通信延迟并记录整理为先验知识，直接带入公式进行对比分析.

3.4 顶点备份方式讨论

图划分是分布式图计算的基础，而划分技术可分为边割与点割两大类. 边割的核心是以顶点为中心进行图划分，即将顶点分配到各计算任务；如果一条边的2个顶点位于不同任务，则该边成为切割边，在迭代计算过程中会引入通信开销；因此在顶点分配时应考虑切割边的规模以优化通信开销，Pregel等系统均以边割方式运行图算法[11]. 点割的核心是以边为中心完成图划分，即将边分配到各计算任务；如果同一个顶点关联的2条边位于不同任务，则该顶点被切分，且多个切分点中会随机选择一个作为主控顶点master而其余作为切分后的从节点mirror存在，在迭代计算过程中的通信仅发生在master与mirror之间. 显然，边分配的过程应尽量减少顶点被切分的概率. PowerGraph等系统以点割方式运行图计算[9].

本文的顶点备份是在边割基础上进行的通信优化. 给定边割的图划分结果，也即顶点在任务间的分布已经确定，备份机制将对每个顶点v(master)的出边进行解析，通过分析其目的顶点在任务间的分布来评估后续计算过程中的通信开销；如果v与任务T_i间的通信过高(即指向T_i中顶点的出边数目超过阈值θ)，则将v中对应的边定向迁移到T_i中并在T_i进行v的备份(mirror).

基于以上描述，本文备份框架与点割方案中，虽然顶点均存在master与mirror的功能角色之分，但在备份的主动性和方向性方面存在区别.

1)备份的主动性. 在基于边割的顶点备份优化框架中，由于顶点(master)分布已经确定，可精确分析“顶点—任务”之间的通信开销并主动决定是否进行出边迁移与顶点备份；而点割方案中，采用启发式规则来指导边的分配并在分配过程中直接(被动)完成顶点切分(以及master和mirror的界定)，由于边分配是动态完成的，系统无法主动分析通信收益以决定是否进行顶点切分.

2)备份的方向性. 在基于边割的顶点备份框架中，由于顶点备份是因迁移出边而引起的，故备份的顶点均作为边的起始点而存在，也即仅将高出度顶点v(master)切分为若干备份顶点(mirror)，当v(master)向其出度邻居广播消息时，可先通过网络将v(master)的值同步到备份顶点(mirror)再由备份顶点进行局部广播，即将消息传递给目的顶点，从而优化通信开销；而在点割方案中，为保证同一个任务上的子图完整性，备份顶点(mirror)既可能作为边的起始点存在，也可能作为边的终止点存在，边的起始点可将发往顶点v的消息首先在其各任务的mirror上进行局部计算以减轻v(master)的处理压力(如PageRank算法中，可基于mirror进行消息的局部累加和操作)，边

的终止点可减少 v(master)向其出度邻居广播消息时的通信开销.

下面分析本文舍弃点割,转而基于边割机制设计顶点备份优化机制的原因.

1)从备份的方向性角度.针对本文关注的多维消息类算法,消息值通常不满足累加特性,即无法将多个消息值通过计算合并为一个消息值(如 PageRank 中的累加求和,以及最短路径计算中的最小值计算),而只能将消息值进行简单串联连接(即本文表 1 中的值连接类算法),此时点割机制中,作为起始点存在的 mirror 不但失去“先局部计算以减轻 v(master)处理压力”的意义,反而引入了 mirror 的存储开销与维护开销. 另一方面,本文相关技术基于以块为中心的 pull 框架实现,其基础框架可保证各顶点发送的消息在发送端实现“能合并尽合并”[10],故即使针对单维值可合并的多维算法(如 MSSP),可以 Combine 合并的方式在发送端实现消息的局部合并,且仅在运行时使用,无需始终维护 mirror.

2)从备份的主动性角度.基于边割的顶点备份可保证被备份的顶点均可带来通信收益,而点割机制由于边分配的动态性,无法保证备份的通信收益. 此处,注意到 PowerLyra[20] 基于 PowerGraph 的点割进行了混合备份优化(hybrid-cut),即通过阈值设定,仅针对高度顶点进行点割而对于低度顶点保持边割. 这与本文对高度顶点进行切分,以最大化减少网络通信开销的目的是一致的. 然而,本文是在边割基础上完成顶点备份,而 PowerLyra 是在点割基础上进行优化. 显然,两者在备份方向性的 2 个角度存在区别. 具体地,从顶层设计层面,本文和 PowerLyra 均针对高度顶点进行切分,这必然涉及到“高度”的衡量标准,即备份阈值θ. 从实现层面,本文的θ作用于顶点 v 指向任务 T_i 上目的顶点的出边规模,而非 PowerLyra 中作用于 v 的全部出边(即出度). 由于出边规模超过阈值即会产生备份,考虑到高出度顶点指向某个具体任务的出边也可能较少,显然本文的作用域更为精确,可确保通信收益. 其次,PowerLyra 并未给出阈值θ的推荐方式,仅以多次重复实验的手工方式选择较优阈值;而本文在第 4 节分析了迁移导致的负载偏斜代价与通信收益,可基于统计信息给出推荐的最优阈值并在 5.4 节通过大量实验验证了相关方案的可行性.

下面通过实例分析,展现本文备份机制的轻量级特点. 假设分布式任务的数目为 3,图 4 给出了一个包含 6 个顶点、9 条边的示例图在 PowerLyra 和本文轻量级备份框架下的备份情况分析. 设定备份阈值θ=3, PowerLyra 以边表的方式并行加载输入图并根据边的源顶点的 Hash 值,即 T_i=$hash$(e_{xy}. x−1)%3,决定该边的分配位置. 然后统计各顶点的出度,如果出度值大于等于 3,则认定为高度顶点,则按照该顶点关联出边的目的顶点重新分配出边,即 T_i=$hash$(e_{xy}. y−1)%3. 这里顶点 v_1 被判定为高度顶点,其出边 e_{12} 与 e_{15} 被迁移到任务 T_2,而 e_{13} 被迁移到任务 T_3. 最后按照备份方向性的讨论,完成顶点备份,即 T_1 中的 v_3, T_2 中的 v_1 以及 T_3 中的 v_1 与 v_2. 然而, T_2 中的 v_1 显然无法进行通信优化,因为 v_1 在该任务上仅有一个目的顶点,备份与否并不能优化通信规模. 这是由于点割方案无法主动控制顶点切分而导致的现象. 相反地,本文轻量级备份以邻接表作为输入,并利用 Range 划分按照字节规模均衡分割、并行加载. 而对于高度顶点的界定,采用“顶点—任务”模式进行主动界定. 此处,只有顶点 v_1 向任务 T_2 进行边 e_{12}, e_{13}, e_{14} 的迁移并备份 v_1,因为 v_1 指向 T_2 的出边数目大于等于阈值θ,从而保证通信收益. 注意到在本文备份机制下,出边被迁移后,任务 T_2 的负载加重,而其中的偏斜程度与阈值的设定相关. 第 4 节将详细讨论阈值的设定问题.

综上,基于本文关注的多维消息算法的巨大内存开销,以及以块为中心的、最新的 pull 系统框架,考虑到点割的维护开销和通信收益的不确定性,本文基于边割的图划分技术,通过顶点备份进行通信再优化. 故本文备份机制的轻量级特点,可总结为 4 点:1)优化的 pull 同步方式可显著减少备份顶点同步过程中的内存开销并与普通消息的 pull 方式统一,便于系统级优化(如容错控制);2)仅按照出边方向进行顶点备份,减少备份开销;3)通过精确控制备份阈值的作用范围,避免无效的冗余备份,保证通信收益;4)提供备份阈值的自动优化计算模型,避免频繁手动测试的阈值选择方式.

4 自适应性能优化模型

本文基于 Range 划分完成边割,而 Range 方法将输入图(由顶点和出边组成)的数据规模进行均等切分,可保证各计算节点负载(即顶点和出边的数量之和)的均衡性. 在此基础上,顶点备份框架在图划分阶段额外引入各任务间顶点的备份和出边的迁入迁出等操作. 考虑到真实图的度分布通常有幂律偏斜特点,备份顶点在各任务间的分布也具有偏斜,且每

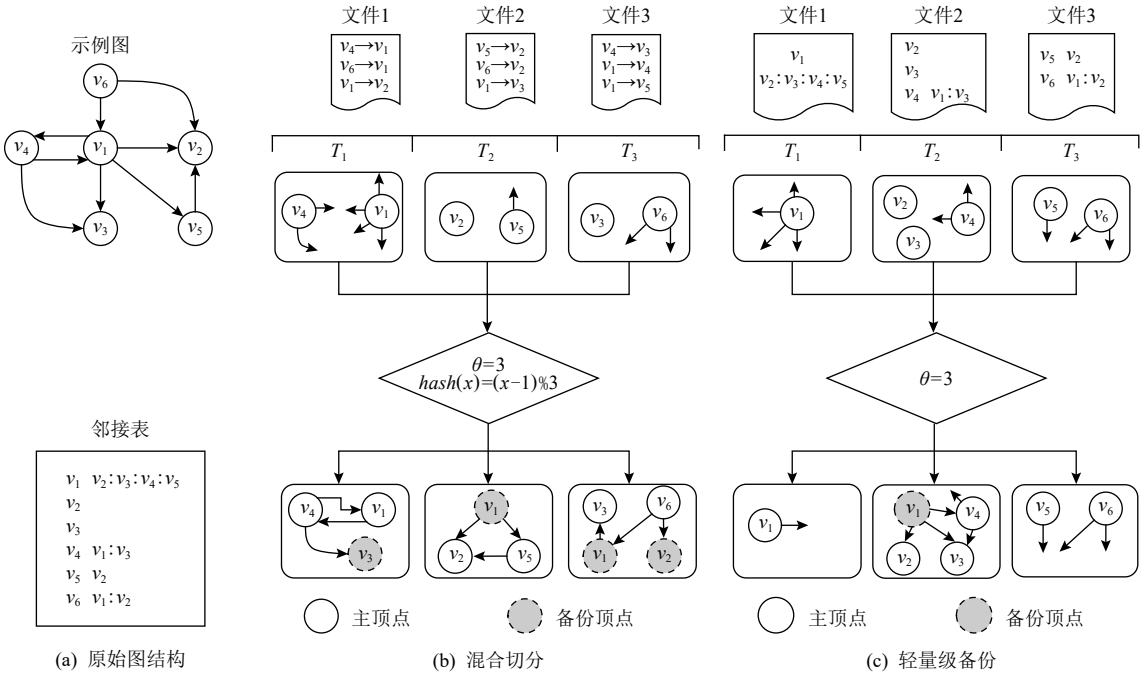


Fig. 4 Comparison of hybrid-cut and lightweight vertex replication

图 4 混合切分与轻量级顶点备份的对比

个任务迁移边的规模不尽相同,这显然破坏了原 Range 划分的负载均衡. 故本文设计的框架对负载均衡方面没有改善,大部分情况下甚至会加重负载偏斜.

在顶点备份机制中,位于任务 T_i 上的顶点 v 是否需要在任务 T_j 上进行备份,取决于其出边是否被迁移. 直观地,如果 v 的邻接表中存在大量指向 T_j 上目的顶点的出边,则边迁移可显著降低通信代价,但同时也会引起 T_i 与 T_j 的负载变化进而影响性能. 因此,需要根据通信收益和负载影响综合考虑,设定出边迁移阈值θ,当指向 T_j 的出边数目超过θ时,证明通信收益可抵消负载变化影响,允许迁移,否则禁止迁移. 显然θ的设定对备份机制的实际性能收益至关重要. 在实际应用场景,可通过多次运行迭代算法手动寻找最优阈值,但这会浪费大量计算资源,可操作性较差. 一种理想的方式是给出θ相关的性能函数,然后自动分析最优阈值以指导实际算法的运行. 本节重点介绍一种基于线下先验知识与线上实时信息相结合的阈值计算模型,其中 4.1 节介绍预测函数,而 4.2 节介绍重要参数的线下与线上获取方式.

4.1 出边迁移阈值优化预测函数

轻量级顶点备份框架的性能预测指标要综合考虑顶点备份后的通信净收益和备份前后各任务负载均衡程度变化导致的水桶效应影响. 给定迁移阈值θ,式(4)给出了性能预测函数的逻辑结构,其中φ_{com}表示顶点备份后的通信净收益,而φ_{load}代表备份前后各任务的负载均衡变化引起的水桶效应影响.

$$\varphi = \varphi_{\text{com}} + \varphi_{\text{load}}. \tag{4}$$

对于通信净收益φ_{com},由第 3 节可知,顶点备份在产生消息通信收益的同时,会引入备份顶点值的同步开销. 其中,消息通信收益取决于顶点备份所产生迁移的出边数量$|\overline{E}|$(E 上面的横杠表示备份)以及沿出边发送的消息字节大小η_{msg},而同步开销则取决于备份顶点的数量$|\overline{V}|$和被同步的顶点值字节大小η_{val}.$|\overline{E}|\eta_{msg}-|\overline{V}|\eta_{\text{val}}$从字节规模角度给出了通信的净收益. 需要注意的是,分布式网络通信的基本流程是首先在发送端进行数据序列化,然后将序列化后的数据通过网络传输到接收端,接收端进行反序列化操作之后即可得到可用数据. 因此,在得到消息总规模和同步数据总规模后,可根据网络传输速率S_{net}和接收端、发送端的序列化、反序列化速率S_{io}来计算净性能收益φ_{com}:

$$\varphi_{\text{com}} = \frac{|\overline{E}|\eta_{msg}-|\overline{V}|\eta_{\text{val}}}{P}\cdot\left(\frac{1}{S_{\text{net}}}+\frac{2}{S_{\text{io}}}\right), \tag{5}$$

其中,P 代表共同参与计算的分布式任务数目,式(5)等号右边第 1 项为分布式环境下的净性能收益;序列化和反序列化需要在发送端和接收端分别执行,因此需要将字节规模乘以系数 2.

对于负载均衡变化导致的水桶效应影响φ_{load},考虑到某个任务 T_i 在向其他任务迁移出边的同时,也在接收其他任务迁入的边数据. 这种迁入迁出会打

破既有图划分结果的均衡性,进而影响负载偏斜程度,导致水桶效应延迟发生变化.分布式环境下,系统处理性能取决于负载最重的任务,因此可用备份前后最重负载的差值作为衡量指标.若备份后的负载均衡状况优于备份前,则负载指标的计算结果为正,对处理性能起正向加速作用;反之,则会降低系统处理速度.φ_{load}的计算方式为

$$\varphi_{\text{load}}=\frac{1}{S_{\text{tpt}}}\cdot\max\{\alpha|V_i|\cdot\eta_{\text{val}}+|E_i|\cdot\eta_{msg}\}-\frac{1}{S_{\text{tpt}}}\cdot\max\left\{\left(\beta\left|\overline{V_j}\right|+\alpha|V_j|\right)\cdot\eta_{\text{val}}+\left(\left|\overline{\Delta E_j}\right|+|E_j|\right)\cdot\eta_{msg}\right\},\tag{6}$$

其中$1\leqslant i\leqslant P$, $1\leqslant j\leqslant P$, $|V_i|$和$|E_i|$分别表示计算任务T_i上分配的子图G_i的顶点数和边数,而$\left|\overline{V_j}\right|$和$\left|\overline{\Delta E_j}\right|$分别表示备份到任务$T_j$上的顶点数和由顶点备份导致的出边迁入迁出变化数.此外,无论是本地顶点还是备份顶点,在计算更新或同步更新时,会产生计算负载,因此分别加入调节因子α和β以调节其相对于边操作的负载.其中,α的值取决于顶点的计算更新以及遍历参与计算更新的接收消息的复杂度;β的值取决于备份顶点的同步更新.显然,α和β的值由算法和数据集共同确定.最后,S_{tpt}为系统吞吐效率,可在给定集群上通过运行标准测试程序获得.

4.2 关键预测参数的获取

根据式(4)~(6),当φ为正值时可提高计算效率,而φ的预测值主要取决于4类参数:1)迁移与备份相关类参数,具体包括备份顶点数$\left|\overline{V}\right|$与$\left|\overline{V_j}\right|$,迁移边数$\left|\overline{E}\right|$与$\left|\overline{\Delta E_j}\right|$,和图划分结束后各任务的子图分布$|V_i|$与$|E_i|$,其中$|V_i|$与$|E_i|$的取值依赖具体的图数据集拓扑结构以及分布式任务数目P,而备份与迁移参数还与备份阈值θ密切相关;2)应用算法类相关参数,主要包括η_{val},η_{msg},其取值由应用层面的图迭代算法决定;3)硬件配置类参数,即S_{net}, S_{io}, S_{tpt},可通过在给定集群上运行标准测试程序获得;4)权重调节因子α和β,可通过分析应用算法复杂度与图数据集的平均出入度计算得到.在上述4类参数中,第3类属于固定常量,只要集群的硬件配置不变,无需反复测试,较易获取和维护;第2类和第4类与具体的应用算法和数据集相关,需要根据用户提交的作业程序实时分析,属于较易获取的线上实时参数;第1类因涉及图拓扑结构以及关键变量θ,难以通过直观的理论分析进行准确估计,因此本节对第1类参数的获取进行详细讨论.

虽然第1类参数难以理论评估,但注意到其只与数据集和集群任务配置相关,而与具体的应用算法无关.考虑到具体领域的图应用通常是根据指定的数据集进行多方位的挖掘分析,如社交网络公司对其运营的社交网络图进行社团聚类、广告推荐以及成员影响力评估等多种业务分析,论文检索系统对学术研究网络进行合作研究团队识别、新研究领域发现以及学界泰斗与新星挖掘等业务分析.这表明,针对一个数据集,通常会从不同角度进行不同类别的应用分析,即多次在同一个数据集上运行不同算法.因此,可对给定的数据集和任务数目配置,通过线下变换θ值统计不同任务上的第1类参数值并保存为先验知识.当需要在指定数据集上运行某种算法时,可依据先验知识和算法相关的实时信息,立即计算出较优的备份阈值,指导轻量级备份框架的运行.

在参数提取阶段,仅需统计各任务的备份顶点数目以及迁移边交换情况,而无需进行具体的迭代计算.因此可直接利用分布式图处理系统的数据加载流程进行逻辑数据统计,而不必进行实际的物理迁移与顶点备份操作,以节省参数提取开销.逻辑统计的另一个优势是,可同时分析多个θ取值下的参数数值,避免针对每个阈值取值进行一次参数提取,进一步压低提取开销.下面通过算法1介绍第1类参数的具体提取过程.

算法1. 备份顶点和迁移边数目统计.

输入:划分给任务T_i的子图G_i,任务数目P,备份阈值数组$\Theta=\{\theta_1,\theta_2,\cdots\}$;

输出:G_i的顶点数$|V_i|$与边数$|E_i|$,各阈值下备份顶点数以及迁出出边数的分布矩阵$\boldsymbol{M}_i$与$\boldsymbol{N}_i$.

① 初始化统计变量$|V_i|$, $|E_i|$, $\boldsymbol{M}_i$和$\boldsymbol{N}_i$;
② while $G_i\neq\varnothing$
③ $adj(v)=loadGraph(G_i)$;
/* 加载1行邻接表并将其从G_i内移除*/
④ $|V_i|$++;
⑤ 初始化目的顶点分布频数统计数组 *dstTid*;
⑥ for($e\in getEdges(adj(v))$)
⑦ $dstTid[getTargetTask(e)]$++;
⑧ $|E_i|$++;
⑨ end for
⑩ for($\theta_j\in\Theta$)
⑪ for(*task* 的 *ID* $k=1$ to P)
⑫ if($dstTid[k]>\theta_j$)
⑬ $updateMatrix(j,k,\boldsymbol{M}_i,\boldsymbol{N}_i)$;
/* 更新备份顶点数和迁移边数*/
⑭ end if
⑮ end for
⑯ end for

⑰ end while

⑱ return $|V_i|$, $|E_i|$, $\boldsymbol{M}_i$, $\boldsymbol{N}_i$.

算法 1 展示了 P 个分布式任务中某个任务 T_i 的运行流程. 该任务对给定的划分子图G_i, 分析各种备份阈值$\Theta=\{\theta_1,\theta_2,\cdots\}$下的顶点备份与迁移边数目等统计信息. 具体地, 通过遍历G_i中的每条邻接表记录, 统计其出边所指向的目的顶点在 P 个任务之间的分布频数并记录在数组 *dstTid* 中(行⑥~⑨); 之后分析不同阈值设定下如θ_j, 是否向对应的任务如 T_k 进行出边迁移以及顶点备份, 如是, 将该统计信息记录在各阈值下备份顶点数以及迁出出边数的分布矩阵$\boldsymbol{M}_i$与$\boldsymbol{N}_i$的第(j,k)位置(行⑩~⑪). 需要注意的是, 此处仅统计分布信息, 而无需对边进行实际物理迁移(行⑬), 因此算法 1 的运行效率较高.

待算法 1 运行结束, 可获得任务 T_i 上的子图 G_i 的顶点数$|V_i|$与边数$|E_i|$, 以及各阈值下备份顶点数以及迁出出边数的分布频数矩阵$\boldsymbol{M}_i$与$\boldsymbol{N}_i$. 随后, 可汇总所有 P 个任务的统计矩阵, 计算每个任务上被备份的顶点数目以及迁入迁出边数目. 具体地, 在阈值θ_j下, 各第 1 类参数的计算方法为: 全体备份顶点数目$\left|\overline{V}\right|=\sum_{i=1}^{P}\sum_{k=1}^{P}M_i[j,k]$, 同理, 全体迁移边数目$\left|\overline{E}\right|=\sum_{i=1}^{P}\sum_{k=1}^{P}N_i[j,k]$; 所有任务在 T_i 上进行备份的备份点数目$\left|\overline{V_i}\right|=\sum_{k=1}^{P}M_k[j,i]$, 而迁入迁出边的变化数目可通过分别统计所有任务 T_i 迁入的边数与 T_i 迁往所有任务的边数来计算, 即$\left|\overline{\Delta E_i}\right|=\sum_{k=1}^{P}\mathbb{Z}_k[j,i]-\sum_{x=1}^{P}N_i[j,x]$. 至此, 通过离线的一次数据加载, 即可获取某个数据集在各个阈值设定下的关键参数, 以提供准确的先验知识. 在此基础上, 当需要在该数据集上运行具体应用算法时, 可结合算法相关的实时信息, 利用 4.1 节提供的模型预测不同阈值的表现, 进而选择最优阈值, 以便在数据加载与图划分阶段直接对邻接表按照 3.2 节的结构进行变换、迁移与备份, 为轻量级顶点备份框架提供关键参数的获取方法.

5 实验结果与分析

5.1 数据集与实验设置

本文在支持完全合并的 HGraph 系统上实现了轻量级顶点备份框架, 可同时支持消息完全合并以及源顶点备份, 在继承 HGraph 系统优势的前提下, 实现备份机制的内存优化和通信性能提升. 为便于区分, 实现轻量级按需备份的系统被称之为 LGraph (light-weight graph). 实验设计方面, 首先在不同数据集上对比轻量级顶点备份与传统 push 备份的内存使用占比(5.2 节), 然后给出轻量级顶点备份与 HGraph 原系统的性能对比与分析(5.3 节), 最后验证自适应性能优化模型的预测分析结果以及备份过程对性能的影响(5.4 节). 应用算法选取表 1 中多维算法 MSSP, SC, SA, 分别作为合并类和连接类的代表. 其中 MSSP 与 SC 的算法逻辑已在 2.2 节中介绍. 而 SA 算法是基于 LPA 设计完成的广告传播模拟算法, 即每个顶点维护自己感兴趣的广告标签列表, 迭代开始后, 各顶点根据入度邻居的广告喜好分布对自己的广告列表进行更新并广播给出度邻居, 其消息值不可合并且消息值需要使用多个 int 数据来表征广告标签. 当涉及运行时间分析时, 由于 SC 与 SA 算法在每步迭代中所有顶点均激活并向所有出度邻居广播消息, 各步的负载相同, 故除非特殊声明, 否则仅汇报一个迭代步的运行时间; 而对于 MSSP, 各步激活顶点规模动态变化, 导致负载也不尽相同, 因此汇报整个算法收敛的总迭代计算时间.

实验集群由 5 台小型服务器组成, 包括 4 个计算节点和 1 个主控节点, 节点配备千兆网卡并使用千兆交换机互联, 实测网络传输性能为 89 MBps①. 主控节点配置 Intel i9-10900K, 3.7 GHz 的 10 核 CPU, 1 TB 固态硬盘, 64 GB 内存; 每个计算节点配置 Intel 至强 E3-2224, 3.5 GHz 的 4 核 CPU, 1 TB 机械硬盘, 32 GB 内存. 实验使用 4 个真实图数据集, 各数据集的具体信息描述如表 3 所示.

Table 3 Description of Real Datasets

表 3 真实数据集描述

数据集	顶点数	出边数	平均出度	文件大小/MB
UK[29]	1 766 010	18 244 650	10	148
LiveJ[30]	4 847 571	69 028 541	14	518
Wiki[31]	6 261 502	150 124 927	23	1 003
EU[32]	11 264 052	386 915 963	34	3 143

实验参数设定方面主要涉及阈值优化模型, 其中网络通信与序列化/反序列化速率为$S_{\text{net}}=89$ MBps, $S_{\text{io}}=507$ MBps, 平均负载吞吐率为$S_{\text{tpt}}=42$ MBps; 另一方面, 负载权重调节因子$\alpha=(\mu_{\text{in}}\cdot\eta_{msg})/S_{\text{upd}}$, $\beta=$

① 网络性能测试使用 iperf-2.0.5 工具

$(\mu_{out} \cdot \eta_{val})/S_{upd}$, 其中$\mu_{in}$与$\mu_{out}$分别为对应数据集的平均入度与平均出度, S_{upd}为顶点更新/同步的CPU处理速度, 实测值为1533MBps. 最后, 对于MSSP类算法的并发源顶点数目设置, 考虑到其合并与备份的通信收益之差与并发粒度成正比, 同时在真实应用环境下通常在硬件允许的前提下采用较大的并发粒度以提高图遍历共享收益, 故在UK和LiveJ数据集上将并发粒度直接设置为平均出入度值; 而对较为稠密的高出入度图Wiki和EU, 将并发源顶点数目设置为平均出入度的2倍, 以强化通信收益.

5.2 备份顶点同步模式对比

本节在4个真实数据集上对比了传统push同步顶点备份方式与按需同步顶点备份方式的内存使用占比情况(即push同步的内存消耗/按需同步的内存消耗)和同步性能, 以证明按需同步顶点备份方式在减少内存资源消耗方面的同时还可以保证相近的同步性能. 表4和表5分别展示了连接(SC)和合并(MSSP)类多维消息算法的对比结果. 由于按块拉取框架的消息按需生成, 因此不同的顶点分块数目决定了按需生成消息的规模. 故测试过程中, 通过将每个任务上的顶点分块数目由2增加到64, 观察2种同步方式的内存消耗变化.

Table 4 Memory Usage of Concatenation Algorithms
表4 值连接类算法内存使用情况

数据集	顶点分块数目					
	2	4	8	16	32	64
UK	1.13	1.26	1.49	1.88	2.44	3.12
LiveJ	1.13	1.26	1.48	1.86	2.42	3.09
Wiki	1.13	1.25	1.47	1.84	2.39	3.07
EU	1.13	1.25	1.47	1.83	2.38	3.05

注: 内存占比=push同步的内存消耗/按需同步的内存消耗.

Table 5 Memory Usage of Combination Algorithms
表5 值合并类算法内存使用情况

数据集	顶点分块数目					
	2	4	8	16	32	64
UK	1.14	1.29	1.58	2.16	3.32	5.64
LiveJ	1.19	1.39	1.78	2.56	4.12	7.24
Wiki	1.30	1.61	2.22	3.43	5.87	10.73
EU	1.45	1.90	2.81	4.62	8.24	15.47

注: 内存占比=push同步的内存消耗/按需同步的内存消耗.

在2类算法中, 按需同步的备份方式均表现出更低的内存使用情况(对比值均大于1). 这是因为按需同步备份方式节省了发送端和接收端的多缓存以及本地消息接收缓存设置. 随着每个任务上的顶点分块数的增加, 每块内部的顶点规模下降, 其接收的消息规模也随之成比例下降, 导致按需生成的消息规模降低, 内存消耗减少; 与此同时, push同步方式的发送与接收端缓存, 只受任务数目的影响, 不随顶点分块的变化而改变. 因此, 随着块规模的增大, 在不同数据集和算法的所有组合测试案例中, 两者的内存消耗对比值均呈现增加趋势. 此外, 对于MSSP, 因不同数据集下其并发源顶点数量的不同, 每条消息的大小也会发生变化, 导致不同数据集下内存收益表现出较大的差异性. 特别地, EU数据集上的MSSP算法并发源顶点数量最多, 需要消耗大量内存, 故在顶点分块为64时2种方案的内存消耗对比最为明显, 此时push同步的内存消耗规模约是本文方法的15倍. 因此, 对于消息规模巨大的多维消息类算法, 采用本文的按需同步方式可有效降低消息传递的规模, 从而减少系统的内存资源消耗.

在同步性能分析方面, 由于备份顶点的同步操作与正常消息值的交换操作紧密耦合, 难以剥离出同步操作的精确时间开销. 考虑到同步方式的不同, 仅影响同步性能而不会影响正常消息的操作效率以及顶点更新效率, 此处采用控制变量法, 即设定其他参数均一致而仅变化备份顶点的同步方式, 然后通过汇报迭代计算过程的运行时间来反映不同同步方式的性能影响. 如表6所示, 通过手动测试不同备份阈值下LGraph的运行时间来确定最优阈值, 然后以最优阈值作为输入, 测试不同同步方式下的运行时间. 这里, npull是未采用3.3节中优先级技术的拉取操作方案而pull是集成优先级技术的方案. 虽然pull方式涉及同步请求发送环节, 但受益于同步字典的冗余消除剪枝作用以及优先级调度, 其同步效率与push方式几近相同(延迟率<2%). 综合表4~6可知, 本文的pull同步方式在不影响同步效率的前提下可显著优化内存使用开销, 从而提升系统在数据处理容量方面的扩展性.

5.3 备份性能分析

本节分别在4个真实数据集上运行3种多维消息类算法, 通过手动测试不同备份阈值下LGraph的运行时间并选择与最优阈值下的性能与无备份机制的HGraph进行对比, 以展现轻量级备份框架的最佳性能收益. 由于算法和数据集本身存在的复杂性和幂律偏斜特性, 每组实验的实际收益各不相同, 图5~7分别展示了对比效果.

Table 6 Comparison of Synchronizing Running Time for Replicated Vertices with pull and push

表 6 pull 与 push 方式下备份顶点的同步运行时间对比 s

数据集	同步方式	SC 算法	SA 算法	MSSP 算法
UK	npull	132.9	47.4	21.3
	pull	122.7	42.4	19.2
	push	121.4	41.7	18.9
LiveJ	npull	253.3	156.7	259.2
	pull	237.6	144.3	244.8
	push	236.5	142.2	239.6
Wiki	npull	586.4	214.8	523.5
	pull	560.3	204.8	508.9
	push	554.9	202.4	503.8
EU	npull	13 630.8	1 063.4	1 098.2
	pull	13 500.8	1 057.8	1 087.1
	push	13 488.5	1 049.7	1 080.4

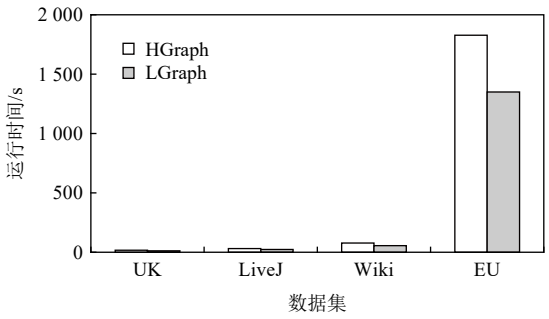


Fig. 5 Running time of SC algorithm on different data sets

图 5 SC 算法在不同数据集上的运行时间

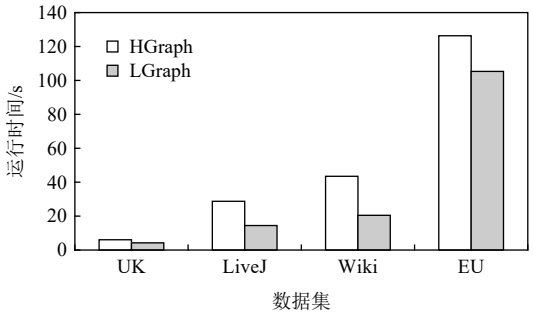


Fig. 6 Running time of SA algorithm on different data sets

图 6 SA 算法在不同数据集上的运行时间

在算法和数据集的各种组合中，LGraph 的计算时间始终低于 HGraph. 特别地，对于连接类算法 SC 和 SA，由于其只能合并目的顶点 *ID*，消息合并收益对整体性能提升并不敏感. 换言之，通信性能的优化主要依靠顶点备份. 此时通过选择较好的备份阈值，可以显著提升整体性能，如 SA 算法在 Wiki 数据集上可以达到 53% 的性能提升. 对于可合并类算法 MSSP，在 UK 和 LiveJ 数据集上，可实现 24% 和 21%

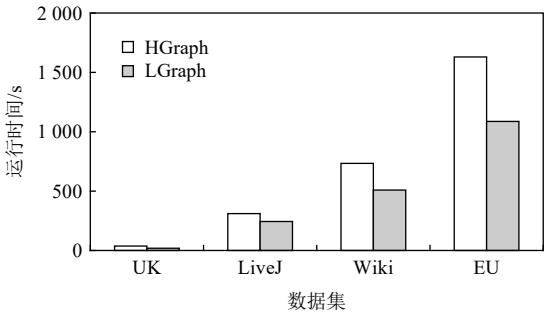


Fig. 7 Running time of MSSP algorithm on different data sets

图 7 MSSP 算法在不同数据集上的运行时间

的性能提升；而对数据集 Wiki 和 EU，由于并发源顶点数目增大，此时性能收益可分别达到 31% 和 33%.

针对各数据集上的不同算法，图 8 和图 9 分别汇报了最优备份阈值对负载和通信的影响，即 4.1 节中分析的因负载偏斜导致的水桶效应φ_{load}以及因备份带来的通信收益φ_{com}. 需要注意的是，实际运行图计算作业时，水桶效应和通信收益同时发生，两者对运行时间的影响紧密耦合，无法精确测量各自的实际影响. 故此处汇报的φ_{load}与φ_{com}均为量化后的理论估算的运行时间(单位为 s)，以展示备份后的负载偏斜代价和通信收益，进而理解本文技术可加速图计算过程的原理.

图 8 中，顶点备份对负载变化的影响是指计算过

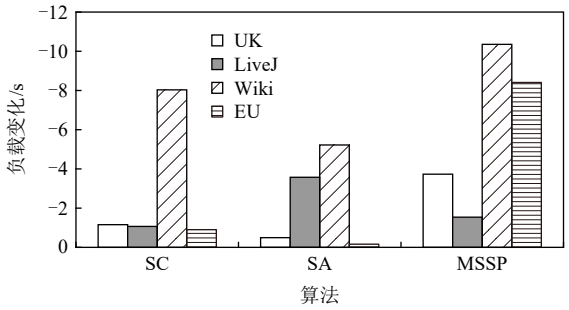


Fig. 8 Analysis on workload variation due to vertex replication

图 8 顶点备份对负载变化的影响分析

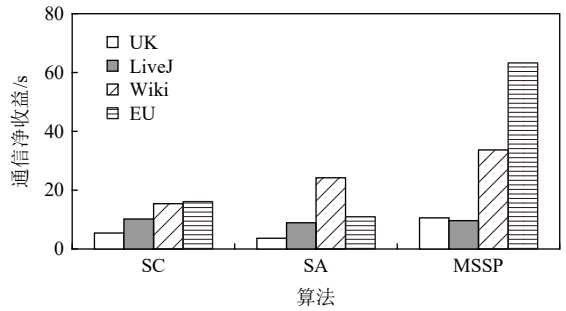


Fig. 9 Analysis on communication net benefit variation due to vertex replication

图 9 顶点备份对通信净收益变化的影响分析

程中水桶效应拖慢的系统运行时间. LGraph 备份后的负载指标计算结果均为负, 即备份后的负载均衡情况劣于备份前, 对加速图计算过程起反向作用. 根据式(6), 负载变化与拓扑结构和消息维度规模密切相关. 从拓扑结构角度来看, Wiki 数据集由于出/入度偏斜指数相差较大, 顶点的备份和边的迁入迁出对其负载影响较大; 而 EU 数据集的高出/入度顶点较多且在各任务间的分布较为均衡, 故备份对负载变化的影响较小. 从消息维度角度来看, MSSP 由于并发源顶点数目多, 导致消息和顶点值的字节数均大于其余 2 个算法, 因此其负载变化幅度通常是最大的. 特别地, 在 LiveJ 数据集上, MSSP 算法的负载变化远小于 SA 算法. 这是由于算法特性导致两者的备份阈值不同. 根据表 7, MSSP 的备份阈值远高于 SA, 导致 MSSP 参与迁移的边以及备份的顶点规模均较少, 故负载变化较少.

Table 7 Comparison of Performance Improvement Between Actual and Predicted Optimal Replication Thresholds

表 7 实际与预测最优备份阈值的性能提升对比

数据集	指标	SC		SA		MSSP	
		实际值	预测值	实际值	预测值	实际值	预测值
UK	最优阈值	0	0	4	1	3	3
	性能提升/%	30/29.5	30/29.5	30/29.0	27/25.9	24/22.5	24/22.5
LiveJ	最优阈值	20	10	1	2	50	40
	性能提升/%	24/22.8	23/22.0	50/47.1	49/47.7	21/20.5	19/18.4
Wiki	最优阈值	15	13	1	11	13	12
	性能提升/%	28/27.1	27/25.8	53/47.5	38/35.0	31/29.6	28/26.8
EU	最优阈值	15	10	30	2	35	40
	性能提升/%	26/26.0	23/22.8	17/16.0	15/13.7	33/32.2	32/31.0

注: 斜杠前后的数值分别为纯迭代计算阶段与加入数据加载阶段的性能提升比.

顶点备份对通信收益变化的影响是指计算过程中顶点备份加快的系统运行时间, 以备份后产生的消息通信收益与引入备份顶点值的同步开销之差作为最终的通信收益指标, 图 9 展示了各算法的通信收益. 对比图 8 和图 9 可以发现, 不同算法在各数据集上的负载变化与通信收益趋势一致, 即高负载偏斜会带来较大的通信收益. 其中, 对于 LiveJ 数据集上的 MSSP 与 SA 算法, 由于 MSSP 算法备份阈值较高, 导致迁移边的规模降低, 故通信收益较少. 综合来看, 通信收益与负载代价之差的变化位于 3.64~63.26 s 之间, 即轻量级顶点备份框架即使引起负载偏斜, 仍能提高图处理的整体性能.

5.4 自适应性能优化模型分析

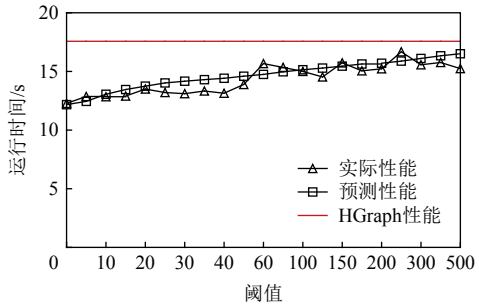
本组实验主要验证备份阈值优化模型的有效性以及所产生的额外开销.

1)模型有效性.自适应优化模型的有效性可通过 2 个方面进行验证, 即公式φ_{load}与φ_{com}对负载偏斜和通信收益估算的准确性以及最优阈值选择的准确性. φ_{load}的验证方式为, 通过在 4 个数据集上运行 SC, SA, MSSP 算法, 首先手动详细测试了不同备份阈值下 LGraph 的实际表现; 对应地, 为便于对比, 将备份阈值优化模型的输出结果(即φ_{load}与φ_{com}理论估算值)累加上无备份的 HGraph 的运行结果, 从而对备份框架的性能进行理论评估. φ_{com}则通过对比手动选择的最优阈值与自适应模型计算的最优阈值及其对应的 LGraph 性能来验证.

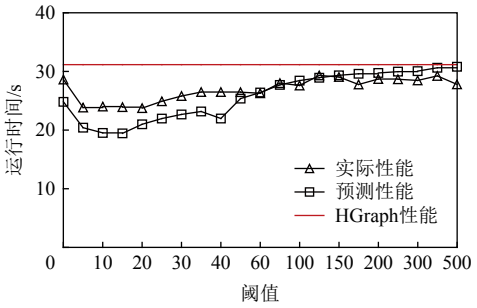
图 10 展示了φ_{load}与φ_{com}的估算准确性验证. 随着阈值的增加, 算法在不同数据集上的运行时间一般呈先下降后上升趋势, 并最终达到甚至超过无备份的 HGraph 运行时间. 算法整体运行时间的变化, 是通信收益与负载偏斜延迟之间相互作用的结果. 前期, 随着阈值增大, 参与备份的顶点(以及迁移的出边)规模减少, 导致通信收益降低, 但同时负载偏斜程度也急剧下降, 因此综合性能收益为正; 后期, 随着阈值持续增大, 通信收益的损失远大于负载偏斜的缓解, 导致综合性能收益为负, 总运行时间呈持续上趋势. 注意到在大部分情况下, 当阈值超过 500 时, 由于指向某一目的任务的最大出度超过 500 的顶点数量极少, 顶点备份产生的通信收益趋于 0, LGraph 的实际迭代性能在此时与 HGraph 相当. 特别地, 对于 EU 数据集上的 SC 算法(图 10(d))和 MSSP 算法(图 10(l)), 由于高出度顶点较多, 当阈值增大时, 仍有大量出边被迁移, 但任务间的负载分布却更为偏斜, 导致通信收益无法抵消负载延迟开销, 使得 LGraph 实际性能甚至不如 HGraph. 此时的阈值分析模型虽不能很好地拟合实际性能表现, 但也可以预测出整

体运行时间呈现上升趋势，从而指导编程人员避免选择较大的阈值. 整体来看，图 10(a)~(l)表明自适应阈值分析模型可较好地拟合实际运行时间的变化趋势，为最优备份阈值选择的准确性提供了保证.

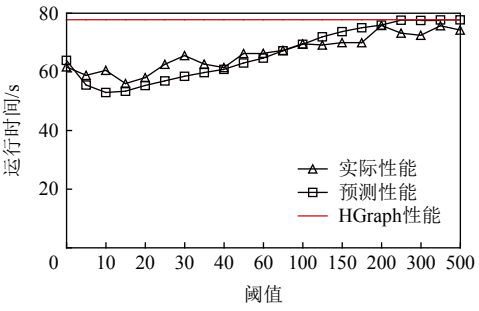
表 7 对比了实际手工测试得到的最优阈值与分析模型计算得到的最优阈值，以验证最优阈值自动选择的准确性. 表 7 同时汇报了累加数据加载与划分开销后顶点备份对整个作业运行时间的优化效果，即“性能提升”斜杠后面的内容. 显然，阈值分析模型在 UK 数据集上的 SC 与 MSSP 算法、LiveJ 数据集上的 SA 算法、Wiki 数据集上的 SC 与 MSSP 算法均可以找到或近似找到最优阈值；对于 LiveJ 数据集上的 SC 与 MSSP 算法、Wiki 上的 SA 算法和 EU 上的 SA 算法，自动计算的最优阈值与实测值相差较大，这是由于收益与延迟开销的博弈接近临界值，对各种参数的取值较为敏感，难以准确预测，但也因此导致最优阈值周围的性能变化幅度较小(见图 10 (b)(j)与图(g)(h))，故即使阈值选择偏差较大，实际的性能收益仍然接近手动选择的最优值.



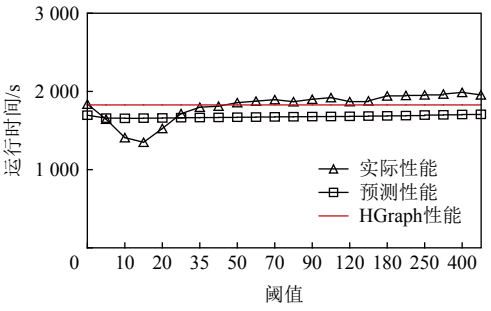
(a) SC-UK



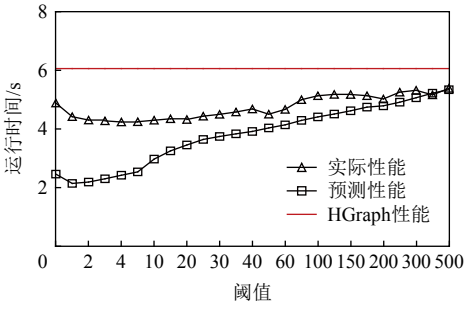
(b) SC-LiveJ



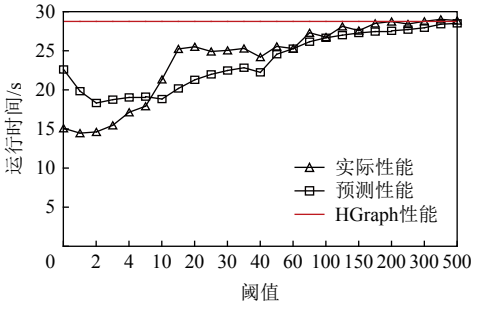
(c) SC-Wiki



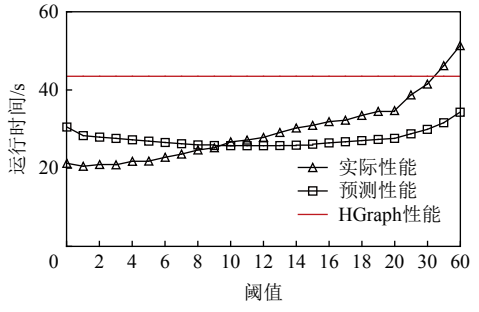
(d) SC-EU



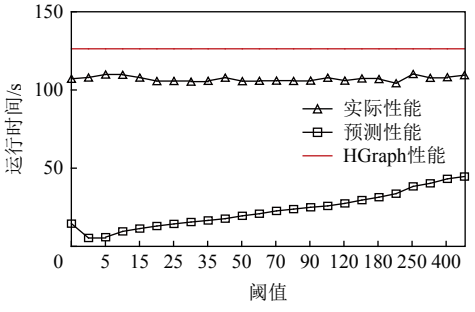
(e) SA-UK



(f) SA-LiveJ



(g) SA-Wiki



(h) SA-EU

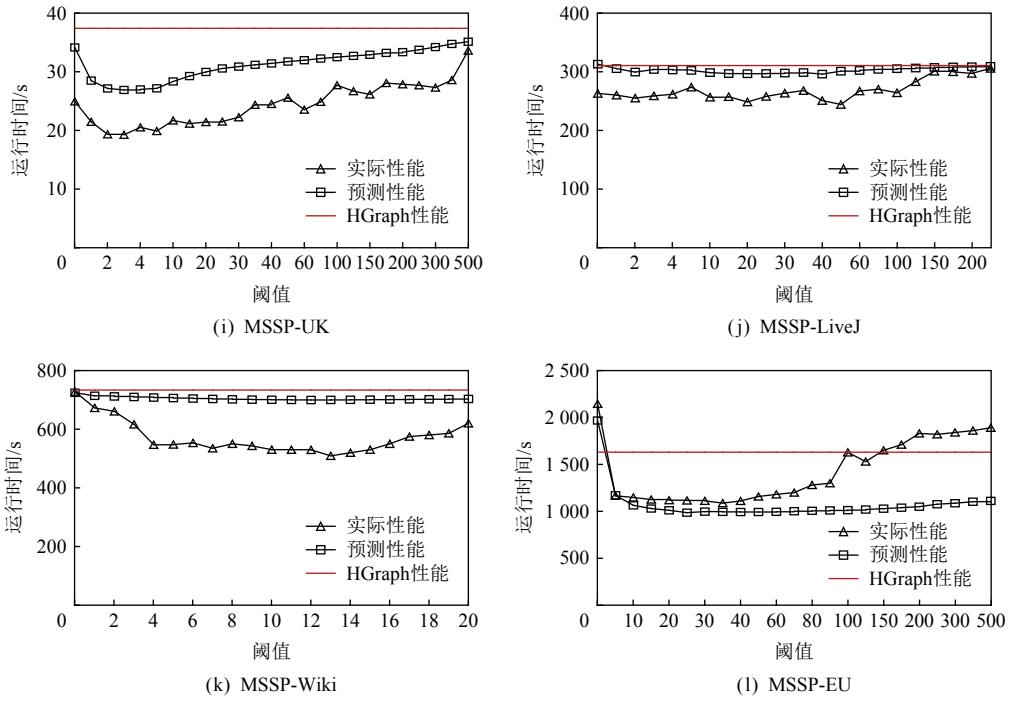


Fig. 10 The actual and predicted performance under different replication thresholds

图 10 不同备份阈值下的实际和预测性能

需要注意的是,对于整个作业的运行时间问题,SC 与 SA 算法均采用 10 步迭代计算的时间之和. 考虑到顶点备份过程内嵌于数据加载与划分阶段,因此,启动顶点备份功能后,系统的加载与划分阶段会引入额外的出边迁移开销. 对比“性能提升”斜杠两边的内容可以看到,即使备份机制在加载划分阶段引入了额外迁移开销,但由于后续迭代过程中产生了巨大的通信收益,前者对综合性能提升百分比的影响十分微小. 以影响最大的 Wiki-SA 组合为例,在手工测试的最优阈值下,性能提升比例由 53% 下降到 47.5%, 仅产生了 5.5% 的影响;而在自适应阈值分析模型下,性能提升比例也仅有 3% 的差距,其综合性能收益仍然十分可观.

2)模型开销.自适应性能优化模型的开销来源于预测所需参数的获取,也即算法 1 展示的第 1 类参数的获取过程. 该过程的核心操作,是在给定的分布式任务数和数据集额外运行一次数据加载,并在加载过程中根据给定的候选阈值数组对不同阈值下的顶点分布以及出边迁移情况进行参数值统计. 图 11 展示了不同阈值粒度(即候选阈值数组长度)下的加载时间开销. 图 12~14 对应列出了 3 种不同算法在不同粒度下阈值选择的准确率. 令θ_s为模型选择的最优阈值,而θ^*为表 7 中汇报的、通过多次手工调试所得的最优阈值. 选择准确性的计算方式为$|\theta_s-\theta^*|/\theta^*$. 结果显示,输入阈值数组的粒度与自适应性能优化模型

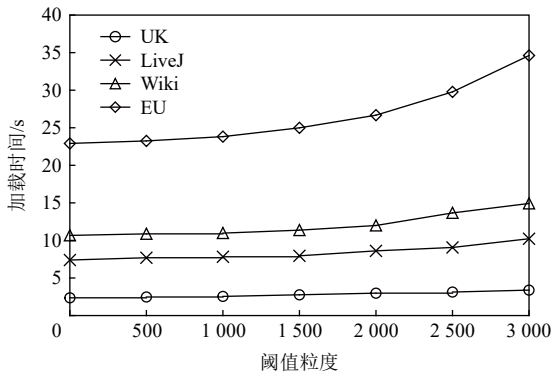


Fig. 11 Latency analysis of loading data under different threshold granularities

图 11 不同阈值粒度下的数据加载延迟分析

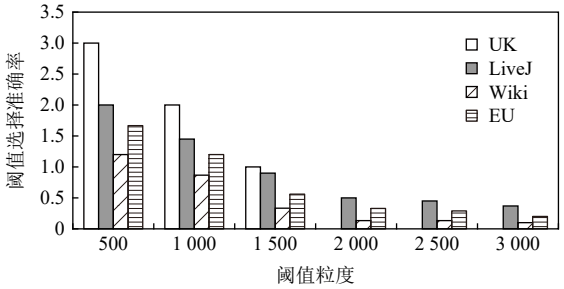


Fig. 12 Accuracy analysis of selecting the optimal threshold under different threshold granularities for SC

图 12 SC 在不同阈值粒度下的最优阈值选择准确率分析

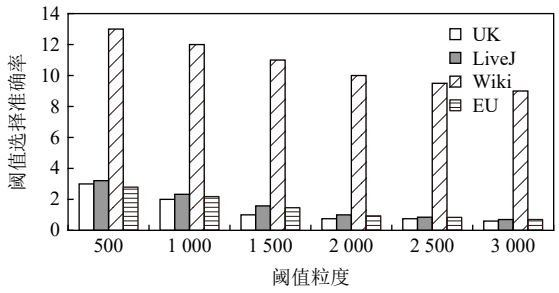


Fig. 13 Accuracy analysis of selecting the optimal threshold under different threshold granularities for SA

图 13 SA 在不同阈值粒度下的最优阈值选择准确率分析

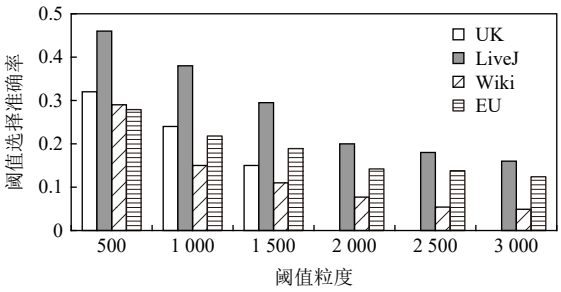


Fig. 14 Accuracy analysis of selecting the optimal threshold under different threshold granularities for MSSP

图 14 MSSP 在不同阈值粒度下的最优阈值选择准确率分析

的统计开销成反比，与最优阈值选择的准确性成正比. 阈值粒度越细化，解析的参数越多，优化模型对最优阈值的预测结果越精细，利于找到最优阈值；反之，最优阈值的选择偏差增大，但参数统计操作减少，加载延迟降低.

综合考虑加载延迟和选择准确性，本文以 2000 为阈值运行优化模型，以 1.12~1.64 倍的延迟获得较高的选择准确率. 此外，考虑到同一个数据集上的不同应用作业可共享参数统计结果，故该加载过程可视为离线操作，其开销不计入实时的作业处理时间.

6 结　论

通信开销一直是制约分布式图处理性能提升的关键因素. 本文从内存和迭代性能上对现有 HGraph 系统进行了改进. 具体地，本文首先对图算法进行分类，指出多维消息类算法对通信和内存的紧迫性要求，并以此为基础在彻底合并系统上引入轻量级顶点备份框架，对系统的内存开销进行优化. 其次，提出了自适应性能优化模型，对顶点参与备份或合并进行定量分析，并对出边偏移阈值进行优化. 大量真实数据集的实验结果表明，轻量级顶点备份框架在内存和执行时间方面，均优于目前最新的处理平台 HGraph，自适应性能优化模型对最优备份阈值的选择也表现出很好的适应性.

作者贡献声明：杜玉洁参与算法构思并负责完成实验方案与论文初稿撰写；王志刚提出了完整的算法框架并修改完成论文终稿；王宁参与了论文的审阅与格式校正；刘芯亦协助完成了相关工作调研与实验数据整理；衣军成完成了实验数据集的收集与格式变换；聂婕与魏志强对论文内容的逻辑布局进行了指导；谷峪与于戈对备份阈值的计算方式提出了指导意见并协助修改论文.

参　考　文　献

[1] Zephoria Digital Marketing. Strategic insight: The Top 10 valuable Facebook statistics – Q2 2021[EB/OL]. (2021-07-28)[2021-12-23]. https://zephoria.com/top-15-valuable-facebook-statistics/

[2] Malewicz G, Austern M H, Bik A J C, et al. Pregel: A system for large-scale graph processing[C] //Proc of the 2010 ACM Int Conf on Management of Data. New York: ACM, 2010: 135−146

[3] Then M, Kaufmann M, Chirigati F, et al. The more the merrier: Efficient multi-source graph traversal[J]. Proceedings of the VLDB Endowment, 2014, 8(4): 449−460

[4] Avery C. Giraph: Large-scale graph processing infrastructure on Hadoop[C/OL] //Proc of the 2011 Conf on Hadoop Summit. 2011 [2021-12-23]. http://giraph.apache.org

[5] Stanton I, Kliot G. Streaming graph partitioning for large distributed graphs[C] //Proc of the 18th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2012: 1222 − 1230

[6] Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs[J]. SIAM Journal on Scientific Computing, 1998, 20(1): 359−392

[7] Salihoglu S, Widom J. GPS: A graph processing system[C/OL] // Proc of the 25th Int Conf on Scientific and Statistical Database Management. New York: ACM, 2013 [2021-12-23]. https://dl.acm.org/doi/abs/10.1145/2484838.2484843

[8] Andreev K, Racke H. Balanced graph partitioning[J]. Theory of Computing Systems, 2006, 39(6): 929−939

[9] Gonzalez J E, Low Y, Gu Haijie, et al. PowerGraph: Distributed graph-parallel computation on natural graphs[C] //Proc of the 10th USENIX Symp on Operating Systems Design and Implementation (OSDI'12). Berkeley, CA: USENIX Association, 2012: 17−30

[10] Wang Zhigang, Gu Yu, Bao Yubin, et al. HGraph: I/O-efficient distributed and iterative graph computing by hybrid pushing/pulling[J]. IEEE Transactions on Knowledge & Data Engineering, 2021, 33: 1973−1987

[11] Yan Da, Cheng James, Lu Yi, et al. Effective techniques for message reduction and load balancing in distributed graph computation

[C]//Proc of the 24th Int Conf on World Wide Web. New York: ACM, 2015: 1307−1317

[12] Wang Xin, Chen Weixue, Yang Yajun, et al. Research on knowledge graph partitioning algorithm: A survey[J]. Chinese Journal of Computers, 2021, 44(1): 235−260 (in Chinese)
(王鑫, 陈蔚雪, 杨雅君, 等. 知识图谱划分算法研究综述[J]. 计算机学报, 2021, 44(1): 235−260)

[13] Catalyurek U V, Aykanat C. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication[J]. IEEE Transactions on Parallel and Distributed Systems, 1999, 10(7): 673−693

[14] Sanders P, Schulz C. Engineering multilevel graph partitioning algorithms[C] //Proc of the 19th Annual European Symp on Algorithms. Berlin: Springer, 2011: 469−480

[15] Wang Zhigang, Gu Yu, Bao Yubin, et al. OnFlyP: An online distributed partition algorithm for large scale graphs based on edge-exchange model[J]. Chinese Journal of Computers, 2015, 38(9): 1838−1851 (in Chinese)
(王志刚, 谷峪, 鲍玉斌, 等. OnFlyP: 基于定向边交换的分布式在线大图划分算法[J]. 计算机学报, 2015, 38(9): 1838−1851)

[16] Leng Fangling, Liu Jinpeng, Wang Zhigang, et al. Edge cluster based large graph partitioning and iteration processing in BSP[J]. Journal of Computer Research and Development, 2015, 52(4): 960−971 (in Chinese)
(冷芳玲, 刘金鹏, 王志刚等. BSP模型下基于边聚簇的大图划分与迭代处理[J]. 计算机研究与发展, 2015, 52(4): 960−971)

[17] Zhang Yu, Zhao Jin, Liao Xiaofei, et al. CGraph: A distributed storage and processing system for concurrent iterative graph analysis jobs[J]. ACM Transactions on Storage, 2019, 15(2): 1−26

[18] Petroni F, Querzoni L, Daudjee K, et al. HDRF: Stream-based partitioning for power-law graphs[C] //Proc of the 24th ACM Int Conf on Information and Knowledge Management. New York: ACM, 2015: 243−252

[19] Tang Jiawu, Zheng Long, Liao Xiaofei, et al. Effective high-level synthesis for high-performance graph processing[J]. Journal of Computer Research and Development, 2021, 58(3): 467−478 (in Chinese)
(汤嘉武, 郑龙, 廖小飞, 等. 面向高性能图计算的高效高层次综合方法[J]. 计算机研究与发展, 2021, 58(3): 467−478)

[20] Chen Rong, Shi Jiaxin, Chen Yanzhe, et al. PowerLyra: Differentiated graph computation and partitioning on skewed graphs[J]. ACM Transactions on Parallel Computing, 2019, 5(3): 1−39

[21] Mayer C, Tariq M A, Mayer R, et al. GraphH: Traffic-aware graph processing[J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(6): 1289−1302

[22] Zhao Yue, Yoshigoe K, Xie Mengjun, et al. L-PowerGraph: A lightweight distributed graph-parallel communication mechanism[J]. The Journal of Supercomputing, 2020, 76(3): 1850−1879

[23] Zhao Yue, Yoshigoe K, Xie Mengjun, et al. LightGraph: Lighten communication in distributed graph-parallel processing[C] //Proc of the 3rd IEEE Int Congress on Big Data. Piscataway, NJ : IEEE, 2014: 717−724

[24] Yan Mingyu, Li Han, Deng Lei, et al. A survey on graph processing accelerators[J]. Journal of Computer Research and Development, 2021, 58(4): 862−887 (in Chinese)
(严明玉, 李涵, 邓磊, 等. 图计算加速架构综述[J]. 计算机研究与发展, 2021, 58(4): 862−887)

[25] Blondel V D, Guillaume J L, Lambiotte R, et al. Fast unfolding of communities in large networks[J]. Journal of Statistical Mechanics: Theory and Experiment, 2008. DOI: 10.1088/1742-5468/2008/10/p10008

[26] Khayyat Z, Awara K, Alonazi A, et al. Mizan: A system for dynamic load balancing in large-scale graph processing[C] //Proc of the 8th ACM European Conf on Computer Systems. New York: ACM, 2013: 169−182

[27] Chang Lijun, Li Wei, Qin Lu, et al. PSCAN: Fast and exact structural graph clustering[J]. IEEE Transactions on Knowledge and Data Engineering, 2017, 29(2): 387−401

[28] Shang Zechao, Yu Jeffrey Xu. Catch the wind: Graph workload balancing on cloud[C] //Proc of the 29th IEEE Int Conf on Data Engineering (ICDE). Piscataway, NJ: IEEE, 2013: 553−564

[29] Laboratory for Web Algorithmics. uk-2014-tpd[EB/OL]. 2014[2021-12-23].https://law.di.unimi.it/webdata/uk-2014-tpd/

[30] Stanford University. LiveJournal social network[EB/OL]. 2009[2021-12-23].https://snap.stanford.edu/data/soc-LiveJournal1.html

[31] Laboratory for Web Algorithmics. enwiki-2021[EB/OL]. 2021[2021-12-23].https://law.di.unimi.it/webdata/enwiki-2021/

[32] Laboratory for Web Algorithmics. eu-2015-host[EB/OL]. 2015[2021-12-23].https://law.di.unimi.it/webdata/eu-2015-host/

Du Yujie, born in 1996. Master candidate. Her main research interests include big data and cloud computing, distributed big graph processing.
杜玉洁, 1996年生. 硕士研究生. 主要研究方向为大数据与云计算、分布式大图处理.

Wang Zhigang, born in 1987. PhD. associate professor, master supervisor. Member of CCF. His main research interests include cloud computing, distributed graph processing, and machine learning.
王志刚, 1987年生. 博士, 副教授, 硕士生导师. CCF会员. 主要研究方向为云计算、分布式图处理、机器学习.

Wang Ning, born in 1988. PhD, lecturer, master supervisor. Member of CCF. Her main research interests include big data management and data privacy protection.
王　宁, 1988年生. 博士, 讲师, 硕士生导师. CCF会员. 主要研究方向为大数据管理、数据隐私保护.

Liu Xinyi, born in 1998. Master candidate. Her main research interests include big data and cloud computing, distributed big graph processing.

刘芯亦, 1998 年生. 硕士研究生. 主要研究方向为大数据与云计算、分布式大图处理.

Yi Juncheng, born in 1987. Master, engineer. His main research interests include big data analysis, data mining and database.

衣军成, 1987 年生. 硕士, 工程师. 主要研究方向为大数据分析、数据挖掘和数据库.

Nie Jie, born in 1986. PhD, associate professor, master supervisor. Member of CCF. Her main research interests include machine learning, image processing, and AI.

聂　婕, 1986 年生. 博士, 副教授, 硕士生导师. CCF 会员. 主要研究方向为机器学习、图形图像处理、人工智能.

Wei Zhiqiang, born in 1969. PhD, professor, PhD supervisor. Member of CCF. His main research interests include image processing, big data analytics, machine learning and cloud computing.

魏志强, 1969 年生. 博士, 教授, 博士生导师. CCF 会员. 主要研究方向为图形图象处理、大数据分析、机器学习和云计算.

Gu Yu, born in 1981. PhD, professor, PhD supervisor. Senior Member of CCF. His main research interests include database technology, distributed system, parallel computing and cloud computing.

谷　峪, 1981 年生. 博士, 教授, 博士生导师. CCF 高级会员. 主要研究方向为数据库技术、分布式系统、并行计算与云计算.

Yu Ge, born in 1962. PhD, professor, PhD supervisor. Fellow of CCF. His main research interests include database theory and technology, distributed system, parallel computing and cloud computing.

于　戈, 1962 年生. 博士, 教授, 博士生导师. CCF 会士. 主要研究方向为数据库理论与技术、分布式系统、并行计算与云计算.