计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(3): 798−811, 2026

基于混合增量计算的流式图并行处理

申 玥[1,2,3] 张 婕[1] 张 园[1] 曹华伟[1,2,3] 安学军[1,2,3] 叶笑春[1]

[1](处理器芯片全国重点实验室(中国科学院计算技术研究所) 北京 100190)

[2](中国科学院大学 北京 100049)

[3](中关村实验室 北京 100094)

(shenyue24b@ict.ac.cn)

Parallel Streaming Graph Processing with Hybrid Incremental Computation

Shen Yue[1,2,3], Zhang Jie[1], Zhang Yuan[1], Cao Huawei[1,2,3], An Xuejun[1,2,3], and Ye Xiaochun[1]

[1] (*State Key Laboratory of Processors* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)

[2] (*University of Chinese Academy of Sciences*, *Beijing* 100049)

[3] (*Zhongguancun Laboratory*, *Beijing* 100094)

Abstract Streaming graphs can effectively model rapidly changing data in real life, and have been widely used in social network analysis, content recommendation, anomaly detection, and other fields. Based on the fact that the two graph snapshots before and after the graph update have a large amount of the same data, incremental computation reduces the amount of memory accesses and computation in the iterations by storing and reusing the historical results, which effectively improves the performance of streaming graph processing. However, the existing solutions for incremental computation optimization of graph algorithms are often limited by graph algorithms that satisfy specific properties, thus it is difficult to apply them to general graph algorithms. In this paper, the incremental computation is further divided into two modes of refinement and recomputation. The similarities and differences between the two modes are analyzed theoretically, and the performance differences between them are tested experimentally with different datasets, algorithms, and update settings. Based on the two modes, the hybrid incremental computation mode is designed to ensure the correctness of switching, and the switching time is predicted accurately by random forest classifiers. Through performance evaluation and switching effect analysis, it is proved that the general graph algorithms can effectively switch under the hybrid mode. Compared with the state-of-the-art solution DZIG, the average speedup is 1.25.

Key words streaming graphs; incremental computation; dynamic graph processing; graph algorithms; parallel processing; graph processing systems

摘 要 流式图能够对现实生活中数据快速变化的场景进行有效建模,在社交网络分析、内容推荐、异常检测等领域得到了广泛应用。基于流式图更新前后的2个图快照具有大量相同数据的事实,增量计算通过对历史计算结果进行存储和复用来降低迭代计算过程中的访存量和计算量,从而有效提升流式图处理的性能。然而,现有对图算法进行增量计算优化的研究往往受限于满足特定性质的图算法,而难以应用于通用图算法。针对通用图算法的增量计算优化问题,将增量计算进一步划分为基于修正和基于重计算

收稿日期: 2025-03-01; **修回日期**: 2025-09-08

基金项目: 国家重点研发计划项目(2023YFB4502305); 北京市自然科学基金项目(4232036)

This work was supported by the National Key Research and Development Program of China (2023YFB4502305) and the Beijing Natural Science Foundation (4232036).

通信作者: 张园(zhangyuan-ams@ict.ac.cn)

的增量计算模式。理论上剖析了二者的异同点，实验上在不同图数据集、图算法和更新场景设置下测试了二者的性能差异。提出了混合增量计算模式，设计了确保切换正确性的算法，并通过随机森林分类器准确地预测切换时机。性能评估和切换效果分析表明通用图算法在混合增量计算模式下能够进行有效切换，并相比先进的流式图处理系统 DZIG 实现了平均 1.25 的加速比。

关键词 流式图；增量计算；动态图处理；图算法；并行处理；图处理系统

中图法分类号 TP311

DOI: 10.7544/issn1000-1239.202550151 **CSTR:** 32373.14.issn1000-1239.202550151

图作为一种非线性数据结构，凭借其对数据的强大建模能力，在各行业领域得到了广泛的应用。其由顶点集和边集构成，顶点和边分别用来表示实体以及实体之间的关系。例如，对于微博社交网络图，用户被表示成顶点，而用户间的关注或评论被表示成边。现实世界中的图数据往往具有动态性，对动态图进行处理可以得到蕴含时间信息的查询结果。例如，在微博社交网络图上执行 PageRank[1] 算法，可以得到微博用户影响力在一段时间内的变化情况，亦可以求得实时的用户影响力。根据上述 2 种查询结果，动态图处理可以被粗粒度地划分为演化图处理（evolving graph processing）和流式图处理（streaming graph processing）两个研究领域[2]。前者处理的是不同时间窗口的多个图快照，目标是得到查询结果随时间的变化情况[3-5]；而后者处理的是一个不断更新的图快照，目标是得到最新的查询结果[6-8]。本文研究的是流式图处理领域的相关理论和技术。

对流式图进行处理需要在图数据变化的过程中持续进行计算，以不断更新查询结果。流式图处理在许多对查询结果实时性要求较高的应用中至关重要，例如异常检测[9]、内容推荐[10] 和社交网络分析[11] 等。工业界和学术界处理流式图的技术主要包括传统计算（traditional computation）和增量计算（incremental computation）[12-13]。传统计算即静态图处理领域采用的计算技术，也称为全计算（full computation）。相比传统计算，增量计算通过存储并复用历史计算结果有效地减少了计算量和访存量[14]，从而具有更优的性能。近年来，针对增量计算的研究主要集中在单调类图算法（在迭代计算过程中顶点状态值单调递减或者递增，如宽度优先搜索[15]（breadth first search, BFS）和单源最短路径[16]（single source shortest path, SSSP）算法）优化[6,17-18] 和支持通用图算法的增量计算[7,19-20] 这 2 个方面。然而，对单调类图算法的专用优化技术不能应用于其他非单调类图算法（如 PageRank 算法），而支持通用图算法的研究主要集中在系统级功能开发和性能优化上。因此，现有方法难以在算法层面有效提升通用图算法的增量计算性能。

我们观察到，根据对受更新影响顶点处理方式的不同，增量计算可以进一步划分为基于修正的增量计算模式和基于重计算的增量计算模式。通过对二者进行理论上的对比分析，我们发现 2 种增量计算模式具有诸多差异。基于修正的增量计算模式通过推送（push）方式[15] 访问出射邻居顶点，从而精确触发受更新影响的顶点修正聚合值和顶点状态值，并且需要原子操作处理数据冲突；而基于重计算的增量计算模式采用拉取（pull）方式[15] 访问所有的入射邻居顶点，并据此为受更新影响的顶点重新计算顶点状态值，在这一过程中没有数据冲突。我们进一步对二者进行实验上的对比分析，发现当数据集顶点规模、平均度数、更新边流大小和图算法改变时，2 种增量计算模式在每轮迭代计算存在着性能差异。因此，在图算法迭代计算过程中，通过选择 2 种增量计算模式中性能更优的模式能够提升增量计算性能。但是，实现该目标具有以下 2 个挑战：

1）如何保证切换的正确性。2 种增量计算模式在迭代计算过程中对邻居顶点的访问方式、更新边流的处理方法以及顶点状态值的计算方式等均不相同，难以在 2 种模式之间搭建桥梁实现正确切换。

2）如何准确预测切换的时机。不准确的切换时机不仅会降低切换带来的性能增益，甚至还会错误地切换到性能较差的模式，最终延长算法的执行时间。

针对上述挑战，本文提出了基于混合增量计算模式的流式图并行处理，主要贡献包括 3 个方面：

1）分别阐述了基于修正和基于重计算的增量计算模式的原理，并进行了实验和理论上的对比分析。

2）提出了混合增量计算模式，能够对 2 种增量计算模式进行正确切换，并通过随机森林分类器实现了准确切换。

3）代码实现了本文提出的理论方法，并进行了实验评估。实验结果表明，混合增量计算模式能够

有效提升增量计算的性能,相比先进的流式图处理系统 DZIG[19] 实现了平均 1.25 的加速比。

1 相关工作

对流式图进行分析可以得到实时的查询结果,国内外学术界和工业界围绕流式图处理领域开展了一系列研究工作,开发了一些具有代表性的经典系统。

早期的相关研究主要集中在通用的数据流处理系统,代表工作有 Naiad[21], Flink[22], Tornado[23],这些系统不局限于处理流式图数据。Naiad[21] 是微软公司在 2013 年提出的分布式数据流处理系统,同时支持批处理和流处理,它的一大特色是采用了差分数据流支持增量计算,实现了对历史数据的复用。Flink[22] 是 2015 年提出的 Apache 开源项目,提供了支持批处理和流处理的应用程序编程接口,并支持运行多种图算法。Tornado[23] 是北京大学在 2016 年基于 Storm[24] 开发的系统,其将迭代计算过程划分为维护近似计算结果的主循环和产生实时计算结果的分支循环,从而通过增量计算加快收敛。这些通用的数据流处理系统没有针对流式图数据进行专门优化。

近年来,部分流式图处理系统针对单调类图算法的特性进行了专门的优化。KickStarter[6] 由加州大学河滨分校在 2017 年提出,其通过对所维护的依赖关系树进行剪枝来实现增量计算。清华大学在 2021 年沿用 KickStarter 的思想开发了 RisGraph[17],其对数据结构、并行模式和吞吐量等进行了系统优化。加州大学河滨分校在 2021 年提出了 Tripoline[18],其提出的图三角形不等式解决了单调类图算法在源顶点改变后难以进行增量计算的问题。华中科技大学在 2022 年和 2023 年分别提出了 GraphFly[25] 和 ACGraph[26], GraphFly 基于依赖树划分流式图并根据时空约束异步地处理分块, ACGraph 则通过分析删增边的依赖关系规范顶点的处理顺序,二者均有效降低了增量计算中的冗余访存量和计算量。这些系统虽然实现了较高的流式图处理性能,但只局限于运行单调类图算法。

另外一些流式图处理系统提供了运行单调类和非单调类图算法的能力。GraphBolt[7] 由西蒙菲莎大学在 2019 年提出,其在初始计算阶段对每轮迭代计算产生的计算结果进行存储,之后在增量计算阶段根据更新边流信息修正所存储的历史计算结果,相比传统计算模式有效地降低了计算量和访存量。为了进一步提升 GraphBolt 的计算性能,西蒙菲莎大学进一步在 2021 年提出了 DZIG[19],其为 GraphBolt 引入了稀疏感知能力,只有当顶点 v 前后 2 轮迭代计算的顶点状态值的变化量超过自定义阈值时,顶点 v 才将该变化量传播(propagate)给它的出射邻居顶点,从而减少修正操作。虽然上述系统支持通用图算法的增量计算,但其仅使用了单一的增量计算模式,并未探究采用其他增量计算模式进一步提升计算性能的可能性。

2 研究背景和预备知识

2.1 流式图处理

图 $G=(V,E)$ 是由顶点和边组成的数据结构,其中 V 和 E 分别表示顶点集和边集。设图 G 中有 n 个顶点和 m 条边,则顶点集和边集可分别表示为 $V=\{v_1,v_2,\cdots,v_n\}$ 和 $E=\{e_1,e_2,\cdots,e_m\}$。

流式图是一种动态图,其结构随着源源不断的更新边流(边的插入或删除)的到来而持续变化。流式图处理包括图更新和图计算 2 个阶段。在图更新阶段,更新边流集合 ΔG 被摄入图 G 中,形成更新后的图 $G'=(V',E')$。在图计算阶段,图算法在更新后的图 G' 上执行,计算出图数据变化后的查询结果。随着源源不断的更新边流的到来,流式图的更新和计算持续交替进行,以得到最新的查询结果。

2.2 基于整体同步并行的以顶点中心的迭代计算模型

整体同步并行(bulk synchronous parallel, BSP)[27] 是一种经典的同步并行计算模型,其将并行计算过程抽象为多个超步(superstep),在每个超步中完成计算、通信和同步 3 个阶段的任务,达到超步数量上限或满足终止条件时停止计算。2010 年,谷歌公司基于 BSP 模型提出了最早的图处理系统 Pregel[28]。此后, BSP 模型因其易编程性在图计算领域被广泛使用。

接下来介绍基于 BSP 的以顶点为中心(vertex-centric)的迭代计算模型。该模型将迭代图算法中的 1 轮迭代计算与 1 个超步相对应,在每轮迭代计算的过程中,以顶点为并行计算单元,各顶点通过访问上一轮迭代计算中其邻居顶点和自身的顶点状态值来完成本轮迭代计算中顶点状态值的计算和更新。由于各顶点的邻居顶点数量不同,故每个顶点的任务量亦不相同,在完成每一轮迭代计算的顶点状态值计算后,参与计算的顶点(即活跃顶点)要进行 1 次同步。

本文参考经典静态图处理框架 Ligra[29],将每轮迭代计算的过程进一步划分为边计算和点计算 2 个

阶段，并通过引入活跃顶点（frontiers）集合过滤掉不需要参与计算的顶点（在每轮迭代计算中只需对活跃顶点进行处理）。在边计算阶段，各顶点根据接收到的活跃的入射邻居顶点信息计算聚合值（aggregation values）；在点计算阶段，各顶点根据聚合值完成顶点状态值的计算和更新。边计算阶段对邻居顶点的访问可以通过以下 2 种方式完成：

1）推送方式。各顶点将其上一轮迭代计算的顶点值推送给出射邻居顶点，其出射邻居顶点被动地完成本轮迭代计算的顶点值的计算和更新。当多个顶点（被分配给多个线程）同时更新同一个出射邻居顶点时，推送方式会造成写冲突。

2）拉取方式。各顶点拉取入射邻居顶点上一轮迭代计算的顶点值，主动地完成本轮迭代计算的自身顶点值的计算和更新。由于只有该顶点自身（被分配给 1 个线程）能更新其顶点值，拉取方式不会造成写冲突。

2.3 传统计算模式

传统计算模式即静态图处理领域采用的计算模式[29-32]，也被用于流式图处理领域[33]。其不对更新前的图G上的历史计算结果进行存储和复用，而是在更新后的图G'上，直接从算法给定的初始值开始进行迭代计算。

设$D_i=\{d_i(v_1),d_i(v_2),\cdots,d_i(v_n)\}$和$G_i=\{g_i(v_1),g_i(v_2),\cdots,g_i(v_n)\}$分别表示$n$个顶点在第$i$轮迭代计算中的顶点状态值和聚合值构成的集合，而$D_0$和$G_0$分别表示初始顶点状态值和初始聚合值构成的集合。例如在 PageRank 算法中，所有顶点的初始顶点状态值为 1，初始聚合值为 0。传统计算模式可以用式（1）表示。

$$D_i=f_{\text{trad}}(D_{i-1}), \tag{1}$$

其中映射f_{trad}由采用了传统计算模式的特定图算法定义。

设$d_i(v)$表示顶点v在第i轮迭代计算结束后的顶点状态值，$N_{\text{in}}(v)$和$N_{\text{out}}(v)$分别表示顶点v的入射和出射邻居构成的集合，F_i表示第i轮迭代计算的活跃顶点集合，则边计算过程可表示为

$$A_i(v)=F_i\cap N_{\text{in}}(v), \tag{2}$$

$$g_i(v)=\mathop{\oplus}_{u\in A_i(v)}(d_{i-1}(u)), \tag{3}$$

其中$A_i(v)$表示顶点v在第i轮迭代计算中活跃的入射邻居顶点，$\oplus$表示聚合算子，$g_i(v)$表示顶点v在第i轮迭代计算的聚合值。对于 PageRank 算法，$g_i(v)=\sum_{u\in A_i(v)}\frac{d_{i-1}(u)}{outDegree(u)}$。

而点计算阶段则包括 2 个步骤，首先顶点v根据聚合值$g_i(v)$计算出新的顶点状态值$d_{\text{new}}(v)$，之后判断是否对上一轮迭代计算的顶点状态值$d_{i-1}(v)$进行更新，该过程可以表示为

$$d_{\text{new}}(v)=\oint g_i(v), \tag{4}$$

$$d_i(v)=upt(d_{i-1}(v),d_{\text{new}}(v)), \tag{5}$$

其中$\oint$表示顶点状态值计算算子，在 PageRank 算法中，$\oint g_i(v)=1-d+d\times g_i(v)$（$d$为阻尼系数）。$upt(\cdot)$表示更新算子，若新的顶点状态值$d_{\text{new}}(v)$满足特定算法的更新条件则更新顶点状态值。在 PageRank 算法中，若$|d_{\text{new}}(v)-d_{i-1}(v)|>\epsilon$（$\epsilon$为更新阈值），则$d_i(v)=d_{\text{new}}(v)$，否则$d_i(v)=d_{i-1}(v)$。本文将代码实现的传统计算模式称为 Tradition 模式。

3 现有增量计算模式的对比分析

在流式图处理中，更新边流的大小（表示为$|\Delta G|$）相比更新前的图大小（表示为$|G|$）占比较小，这意味着更新前后的 2 个图快照中存在大量相同的顶点和边[14]。因此，当采用传统计算模式对更新前后的 2 个图快照进行处理时，在边计算阶段会访问大量相同的邻居顶点并计算相同的聚合值，而在点计算阶段则会计算相同的顶点状态值。与传统计算模式不同，增量计算模式在处理更新前的图快照时对产生的计算结果进行了存储，而在处理更新后的图快照时则复用历史计算结果，从而避免了冗余的计算和访存操作。

根据对受更新影响的顶点的处理方式的不同，本文将增量计算进一步划分为基于修正的增量计算模式和基于重计算的增量计算模式。

3.1 基于修正的增量计算模式

基于修正的增量计算模式被大多数先进的流式图处理系统采用[6-7,17-20]，其在图G上进行迭代计算的过程中，对每轮迭代计算的历史计算结果进行存储，之后在更新后的图G'上进行迭代计算时，通过修正所存储的历史计算结果得到最新的计算结果。通过修正，该模式的计算结果与采用传统计算模式得到的计算结果完全一致。

设$D_i^{\text{old}}=\{d_i^{\text{old}}(v_1),d_i^{\text{old}}(v_2),\cdots,d_i^{\text{old}}(v_n)\}$和$G_i^{\text{old}}=\{g_i^{\text{old}}(v_1),g_i^{\text{old}}(v_2),\cdots,g_i^{\text{old}}(v_n)\}$分别表示$n$个顶点在历史第$i$轮迭代计算中的顶点状态值集合和聚合值构成的集合。基于修正的增量计算模式可以用式（6）表达。

$$(D_i,G_i)=f_{\text{refine}}(D_i^{\text{old}},G_i^{\text{old}},D_{i-1}), \tag{6}$$

其中映射f_{refine}由采用了基于修正的增量计算模式的特定图算法定义。下面对边计算和点计算阶段进行介绍。

在边计算阶段,受更新边流集合ΔG影响的顶点(对于 PageRank 算法,顶点状态值或出度发生改变),必须作为活跃顶点参与边计算,并以推送方式触发出射邻居顶点修正历史顶点状态值,不受更新影响的其他顶点则不需要参与边计算。设F_i^{refine}表示基于修正的增量计算模式在第i轮迭代计算的活跃顶点集合,T_i表示第i轮迭代计算中须要修正顶点状态值的顶点构成的集合,则有

$$T_i=\{v|(u,v)\in E',u\in F_i^{\text{refine}}\}。\tag{7}$$

设$R_i(v)$表示第i轮迭代计算中触发顶点v进行边计算的入射邻居集合,则边计算阶段表示为

$$\delta_i(v)=\bigoplus_{u\in R_i(v)}(\delta_i(u,v)),v\in T_i,\tag{8}$$

$$g_i(v)=g_i^{\text{old}}(v)\uplus\delta_i(v),v\in T_i。\tag{9}$$

在式(8)中,$\delta_i(v)$表示第i轮迭代计算中顶点v的聚合值变化量,$\delta_i(u,v)$表示第i轮迭代计算中顶点u对顶点v的聚合值变化量的贡献,$\oplus$为修正聚合算子。在 PageRank 算法中,$\delta_i(v)=\sum_{u\in R_i(v)}\delta_i(u,v)$。在式(9)中,$\uplus$为对历史聚合值进行修正的算子。对于 PageRank 算法,$g_i(v)=g_i^{\text{old}}(v)+\delta_i(v)$。

接下来进入点计算阶段,只有集合T_i中的顶点才需进行新的顶点状态值的计算和顶点状态值的更新,而对不属于T_i的顶点,由于其顶点状态值未受更新边流影响而改变,故直接复用历史顶点状态值。该过程可以表示为

$$d_i(v)=\begin{cases}upt(d_{i-1}(v),\biguplus g_i(v)), & v\in T_i,\\ d_i^{\text{old}}(v), & v\notin T_i。\end{cases}\tag{10}$$

图 1(b)给出了 PageRank 算法在基于修正的增量计算模式下的迭代计算过程示意图,图 1(a)是被处理的流式图示例,图G'由图G插入一条边(v_0,v_1)得到。在第 1 轮迭代计算中,顶点v_0被激活,触发其出射邻居顶点v_1和v_3来修正历史聚合值并计算顶点状态值。在第 2 轮迭代计算中,顶点v_0,v_1,v_3被激活,分别触发它们的出射邻居顶点(顶点v_0的出射邻居顶点为v_1和v_3,顶点v_1没有出射邻居顶点,顶点v_3的出射邻居顶点为v_1和v_4)来修正历史聚合值并计算顶点状态值。在第 3 轮迭代计算中,被激活的顶点为v_0,v_1,v_3,v_4,须要修正顶点状态值的顶点为v_1,v_3,v_4。

3.2 基于重计算的增量计算模式

基于重计算的增量计算模式[8]也是通过对历史计算结果进行存储和复用来进行增量计算,但与基于修正的增量计算模式不同,其不对历史计算结果进行修正,而是直接触发受更新影响而顶点状态值改变的顶点进行重新计算,其他顶点则直接复用历史计算结果。基于重计算的增量计算模式可以用式(11)表示。

$$(D_i,G_i)=f_{\text{recpt}}(D_i^{\text{old}},G_i^{\text{old}},D_{i-1}),\tag{11}$$

其中映射f_{recpt}由采用了基于重计算的增量计算模式的特定图算法定义。

不论是基于修正还是基于重计算的增量计算模式,它们的目标都是对受更新影响而导致顶点状态值变化的顶点进行识别和计算,以得到正确的顶点状态值。因此,基于重计算的增量计算模式中第i轮迭代计算须要参与重新计算的顶点构成的集合,即为基于修正的计算模式中第i轮迭代计算须要修正顶点状态值的顶点构成的集合T_i。在边计算阶段,T_i中的顶点通过拉取方式访问其所有入射邻居的第$i-1$轮迭代计算的顶点状态值来计算聚合值。设基于重计算的增量计算模式在第i轮迭代计算的活跃顶点集合为F_i^{recpt},有

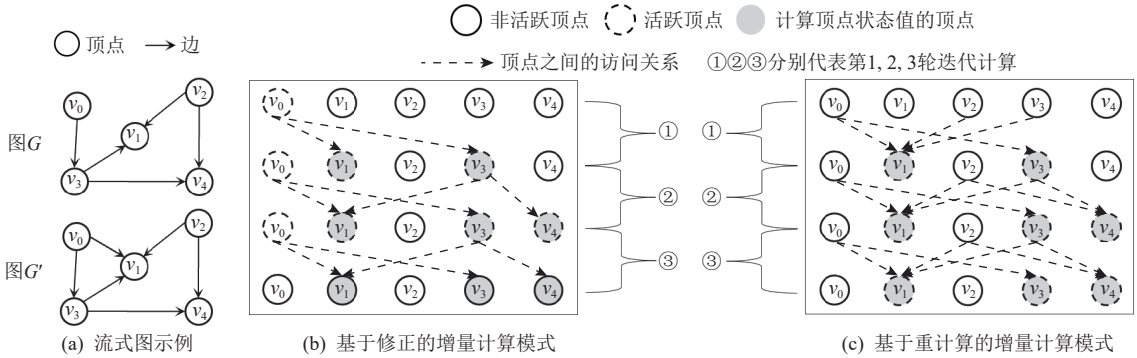


Fig. 1 Illustration of the iterative computation process of PageRank algorithm

图 1 PageRank 算法的迭代计算过程示意图

$$F_i^{\text{recpt}} = T_i \text{。} \tag{12}$$

边计算过程可以表示为

$$g_i(v) = \bigoplus_{u \in N_{\text{in}}(v)} (d_{i-1}(u)), v \in T_i \text{。} \tag{13}$$

在点计算阶段，集合T_i中的顶点计算新的顶点状态值并更新顶点状态值，而不属于集合T_i的顶点，则直接复用历史顶点状态值。该计算过程与基于修正的增量计算模式相同，可以表示为式(10)。

图 1(c)给出了 PageRank 算法在基于重计算的增量计算模式下的迭代计算过程示意图。在第 1 轮迭代计算中，顶点v_1和v_3被激活，它们分别通过访问其入射邻居顶点(顶点v_1的入射邻居顶点为v_0，v_2，v_3，顶点v_3的入射邻居顶点为v_0)重新计算聚合值和顶点状态值。在第 2 轮迭代计算中，顶点v_1，v_3，v_4被激活，分别通过访问它们的入射邻居(顶点v_4的入射邻居顶点为v_2和v_3)来重新计算聚合值和顶点状态值。第 3 轮迭代计算则与第 2 轮迭代计算的过程相同。

3.3 理论对比分析

下面分别在边计算和点计算 2 个阶段从理论上对基于修正和基于重计算的增量计算模式进行对比分析。

1)边计算阶段

由式(7)~(9)可知，对于基于修正的增量计算模式，在第i轮迭代计算的活跃顶点集合F_i^{refine}中的顶点u以推送方式精确触发其出射邻居顶点v来修正历史聚合值$g_i^{\text{old}}(v)$。这意味在第i轮迭代计算中，基于修正的增量计算模式对边的访问量为$|\{(u,v) \in E' | u \in F_i^{\text{refine}}\}|$。而由式(12)(13)可知，对于基于重计算的增量计算模式，在第i轮迭代计算的活跃顶点集合F_i^{recpt}(即集合T_i)中的顶点v以拉取方式访问所有的入射邻居顶点来重新计算聚合值$g_i(v)$。这表明在第i轮迭代计算中，基于重计算的增量计算模式对边的访问量等于集合T_i中所有顶点的入射邻居数量的总和(即$\sum_{v \in T_i} |N_{\text{in}}(v)|$)。因此，基于重计算的增量计算模式相比基于修正的增量计算模式对边的访问量更多以及涉及更多的访存操作，带来更大的访存开销。以图 1 为例，在第 1 轮迭代计算中，对于基于修正的增量计算模式，活跃顶点v_0只访问了边(v_0,v_1)和(v_0,v_3)；而对于基于重计算的增量计算模式，活跃顶点v_1访问了边(v_0,v_1)，(v_2,v_1)，(v_3,v_1)，而活跃顶点v_3则访问了边(v_0,v_3)。

此外，由于基于修正的增量计算模式采用了推送模式来访问出射邻居，故需要采用原子操作来解决数据竞争的问题；而基于重计算的增量计算模式采用了拉取模式来访问入射邻居，不会产生数据竞争，故不需要原子操作。由于原子操作需要一定的时间开销，所以当计算量相同时，基于修正的增量计算模式相比基于重计算的增量计算模式需要更多的计算时间。因此，基于修正和基于重计算的增量计算模式的性能可能在不同图数据集和图算法场景下呈现不同的性能优劣情况。

2)点计算阶段

从 3.1 节和 3.2 节的分析可知，基于修正和基于重计算的增量计算模式的目标均是为受更新影响而顶点状态值改变的顶点(即集合T_i中的顶点)计算新的顶点状态值。由于受更新影响而改变顶点状态值的顶点是由图数据集、更新边流和图算法性质决定的，而与增量计算的模式无关，所以基于修正的和基于重计算的增量计算模式在点计算阶段需要进行顶点状态值计算的顶点数量是相等的。例如，在图 1 中，基于修正和基于重计算的增量计算模式在每一轮迭代中进行顶点状态值计算的顶点相同。因此，在点计算阶段，基于修正和基于重计算的增量计算模式的性能差异较小。

3.4 实验对比分析

综合 3.3 节的分析，我们选择顶点数量、平均度数、更新边流大小和图算法类型这 4 个典型特征作为实验参数，分析不同参数设置情况下 2 种增量计算模式的性能差异。本文将代码实现的基于修正和基于重计算的增量计算模式分别称为 IncRefine 和 IncRecpt 模式。我们采用了 PageRank[1]，Label Propagation[34]，BFS[15]，SSSP[16] 这 4 种典型的图算法，在 RMAT[35] 生成图数据集上进行对比实验，5.1 节对详细的实验设置进行了介绍。

在 RMAT24_16 数据集上，我们设置更新边流的大小$|\Delta G|$分别为 10 和10^7，测试了上述 4 种图算法在 IncRefine 和 IncRecpt 模式下每轮迭代计算的执行时间。图 2(a)给出了 PageRank 算法的实验结果，从中可以看出，当$|\Delta G| = 10$时，IncRefine 模式的性能全过程优于 IncRecpt 模式；当$|\Delta G| = 10^7$时，2 种增量计算模式的性能接近。图 2(b)给出了 Label Propagation 算法的实验结果，从中可以看出，当$|\Delta G| = 10$时，在前 3 轮迭代计算中整体上 IncRefine 模式的性能更优，但在第 4 轮迭代计算以后 IncRecpt 模式的性能更优；当$|\Delta G| = 10^7$时，在第 1 轮迭代计算中 IncRefine 模式的性能更优，但在之后的迭代计算中 IncRecpt 模式的性能更优。图 2(c)给出了 BFS 算法的实验结果，从图中可以看出，当$|\Delta G| = 10$时，在第 2~5 轮迭代计算

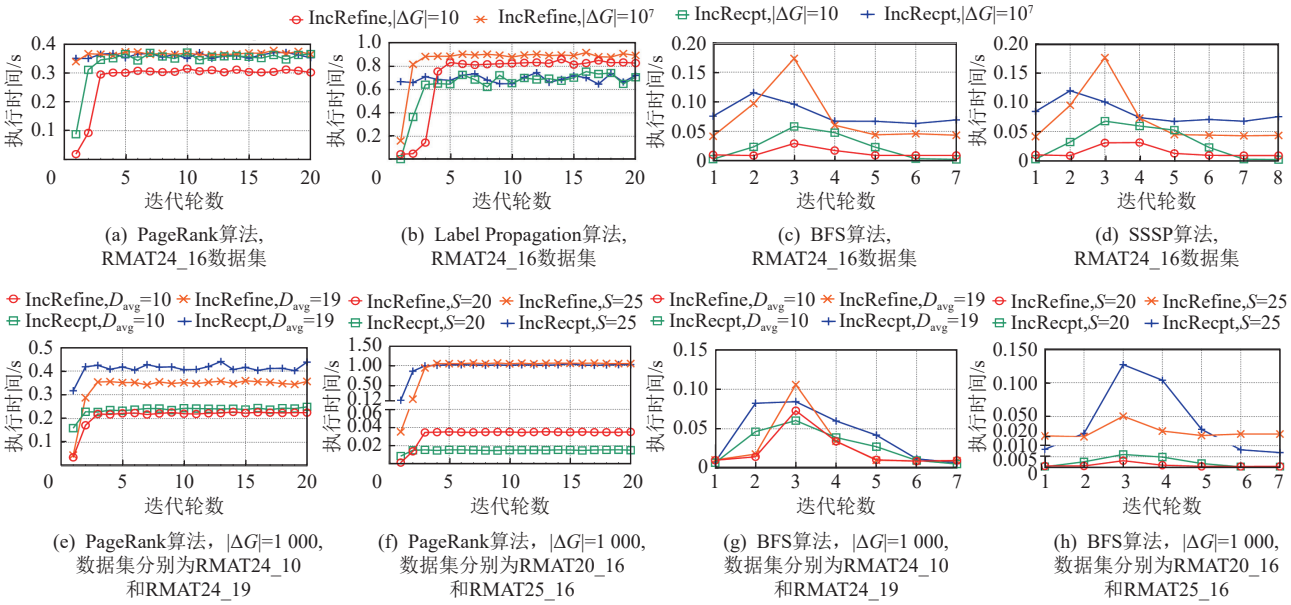


Fig. 2 Experimental comparative analysis of incremental computing models based on correction and recalculation

图 2 基于修正和重计算的增量计算模式的实验对比分析

中 IncRefine 模式的性能更优，在第 1, 6, 7 轮迭代计算中 2 种模式的性能接近；当 $|\Delta G| = 10^7$ 时，在第 3 轮迭代计算中 IncRecpt 模式的性能更优，但在其他轮迭代计算中 IncRefine 模式的性能更优。图 2(d)给出了 SSSP 算法的实验结果，其表现出了和 BFS 算法相近的趋势。总体来说，不同图算法在迭代计算过程中的 2 种增量计算模式呈现出不同的性能优劣情况，这是因为 2 种增量计算模式采用的算子是由特定图算法定义的(例如聚合算子和顶点状态值计算算子)。而当更新边流大小 $|\Delta G|$ 变化时，2 种增量计算模式的性能也呈现出差异，这是因为 $|\Delta G|$ 直接影响了每一轮迭代中需要计算顶点状态值的顶点数量 $|T_i|$，由 3.3 节的理论分析可知，这会导致 2 种增量计算模式的访存开销和计算开销发生变化，从而影响算法性能。

为了进一步探究图数据集的大小对实验结果的影响，我们在 RMAT24_10, RMAT24_19, RMAT20_16, RMAT25_16 这 4 个数据集上，分别测试了 4 种图算法在 2 种增量计算模式下每轮迭代计算的执行时间，其中 $|\Delta G|$ 的大小设置为 10^3。限于篇幅，我们只展示了 PageRank 和 BFS 算法的实验结果。图 2(e)(f)给出了 PageRank 算法在 4 个大小不同的图数据集上的执行结果。从图 2(e)中可以看出，在 RMAT24_10 和 RMAT24_19 数据集上，IncRefine 模式的性能整体上优于 IncRecpt 模式。从图 2(f)中可以看出，在 RMAT20_16 数据集上，除了前 2 轮迭代计算以外，IncRecpt 模式的性能优于 IncRefine 模式；在 RMAT25_16 数据集上，在前 2 轮迭代计算中 IncRefine 模式的性能更优，之后 2 种模式的性能则接近。图 2(g)(h)给出了 BFS 算法的实验结果。从图 2(g)中可以看出，对于 RMAT24_10 和 RMAT24_19 数据集，在迭代计算前期和后期 IncRefine 模式整体上性能更优，在第 3 轮迭代计算中 IncRecpt 模式的性能更优。从图 2(h)中可以看出，在 RMAT20_16 和 RMAT25_16 数据集上，在迭代计算前期和后期 IncRecpt 模式的性能整体上更优，在迭代计算中期 IncRefine 模式的性能更优。综合来看，在不同的图数据集规模下，2 种增量计算模式呈现出不同的性能优劣情况。当平均度数和 $|\Delta G|$ 相等时，顶点数量更多的图数据集带来了更长的图算法执行时间，这是因为需要计算顶点状态值的顶点数量 $|T_i|$ 增多，由 3.3 节的理论分析可知，这会间接导致 2 种增量计算模式呈现不同的性能优劣情况。而当图数据集的顶点数量和 $|\Delta G|$ 相等时，平均度数的增加很可能会导致在边计算阶段 IncRecpt 模式相比 IncRefine 模式的边访问量增加得更多，这也会导致 2 种增量计算模式呈现不同的性能优劣情况。

通过实验对比分析可以看出，随着迭代计算的进行，不同的顶点数量、平均度数、更新边流大小和图算法类型都会影响 2 种增量计算模式的性能。在迭代计算过程中较优的增量计算模式不是固定不变的。

4 混合增量计算模式

本节提出并代码实现了混合增量计算模式(下文称为 IncHybrid 模式)，首先给出在 2 种增量计算模

式之间正确切换的算法，然后介绍基于随机森林分类器的切换机制。

4.1 切换正确性保证

我们通过在2种增量计算模式之间传递迭代计算所需的活跃顶点集合来保证切换的正确性，即由其中一种增量计算模式在第i轮迭代计算中的活跃顶点集合求得另一种增量计算模式在第$i+1$轮迭代计算中的活跃顶点集合。

1）IncRefine模式切换到IncRecpt模式

由式(7)(12)可知，IncRefine模式在第i轮迭代计算的活跃顶点集合F_i^{refine}中所有顶点的出射邻居构成的集合即为IncRecpt模式在第i轮迭代计算的活跃顶点集合F_i^{recpt}。因此，我们先通过活跃顶点集合F_i^{refine}求出活跃顶点集合F_i^{recpt}，再根据活跃顶点集合F_i^{recpt}求出活跃顶点集合F_{i+1}^{recpt}。具体过程如算法1所示。首先并行地遍历活跃顶点集合F_i^{refine}中的顶点并将其出射邻居顶点放入活跃顶点集合F_i^{recpt}中(行①~⑤)，活跃顶点集合F_i^{recpt}中的顶点均被激活为下一轮迭代计算的活跃顶点(行⑥)，之后并行地遍历活跃顶点集合F_i^{recpt}中的顶点并将其出射邻居顶点放入活跃顶点集合F_{i+1}^{recpt}中(行⑦~⑪)。

算法1. IncRefine模式切换到IncRecpt模式。

输入：IncRefine模式在第i轮迭代计算的活跃点集F_i^{refine}；

输出：IncRecpt模式在第$i+1$轮迭代计算的活跃点集F_i^{recpt}。

① parallel-for $u \in F_i^{\text{refine}}$ do
② parallel-for $v \in N_{\text{out}}(u)$ do
③ 将顶点v放入集合F_i^{recpt}中；
④ end parallel-for
⑤ end parallel-for
⑥ $F_{i+1}^{\text{recpt}} \leftarrow F_i^{\text{recpt}}$；
⑦ parallel-for $u \in F_i^{\text{recpt}}$ do
⑧ parallel-for $v \in N_{\text{out}}(u)$ do
⑨ 将顶点v放入集合F_{i+1}^{recpt}中；
⑩ end parallel-for
⑪ end parallel-for

下面以图1中PageRank算法在IncRefine模式下的第1轮迭代计算切换到IncRecpt模式下的第2轮迭代计算为例介绍该切换过程。在IncRefine模式下的第1轮迭代计算中(图1(b))，$F_i^{\text{refine}}=\{v_0\}$，将其输入算法1中，输出为$F_{i+1}^{\text{recpt}}=\{v_1,v_3,v_4\}$(对应图1(c)第2轮迭代计算中的活跃顶点)。

2）IncRecpt模式切换到IncRefine模式

IncRecpt模式在第i轮迭代计算的活跃顶点集合F_i^{recpt}中的所有顶点均属于IncRefine模式在第$i+1$轮迭代计算的活跃顶点集合F_{i+1}^{refine}。此外，在IncRefine模式中，受更新边流ΔG直接影响的顶点也要作为活跃顶点参与每轮迭代计算。具体过程如算法2所示。首先，活跃顶点集合F_i^{recpt}中的顶点均被激活为IncRefine模式在第$i+1$轮迭代计算的活跃顶点(行①)，然后并行地遍历更新边流集合ΔG中的边并根据特定图算法定义的影响算子$affect(\cdot)$判断是否激活该边的源顶点和目的顶点(行②~⑨)。影响算子$affect(\cdot)$负责判断顶点是否受更新边流ΔG直接影响。例如，在PageRank算法中，影响算子$affect(\cdot)$的功能为判断某个顶点的出度是否发生改变，若改变则输出true，否则输出false。

算法2. IncRecpt模式切换到IncRefine模式。

输入：IncRecpt模式在第i轮迭代计算的活跃点集F_i^{recpt}；

输出：IncRefine模式在第$i+1$轮迭代计算的活跃点集F_{i+1}^{refine}。

① $F_{i+1}^{\text{refine}} \leftarrow F_i^{\text{recpt}}$；
② parallel-for $(u,v) \in \Delta G$ do
③ if $affect(u)$ = true then
④ 将顶点u放入集合F_{i+1}^{refine}中；
⑤ end if
⑥ if $affect(v)$ = true then
⑦ 将顶点v放入集合F_{i+1}^{refine}中；
⑧ end if
⑨ end parallel-for

下面以图1中PageRank算法在IncRecpt模式下的第2轮迭代计算切换到IncRefine模式下的第3轮迭代计算为例介绍该切换过程。在IncRecpt模式下的第2轮迭代计算中(图1(c))，$F_i^{\text{recpt}}=\{v_1,v_3,v_4\}$，将其输入算法2中，输出为$F_{i+1}^{\text{refine}}=\{v_0,v_1,v_3,v_4\}$(对应图1(b)第3轮迭代计算中的活跃顶点)。

4.2 基于随机森林分类器的切换机制

本文采用随机森林分类器[36]预测2种增量计算模式的切换时机。随机森林分类器由多棵决策树构成，通过每棵决策树投票得出最终的预测结果，其具有较高的预测准确率和鲁棒性，且不容易出现过拟合问题。

预测2种增量计算模式的切换时机通常可以采用2种方式：1)在迭代计算之前对切换时机进行预测(在哪一轮迭代计算进行切换)；2)在迭代计算过

程中对切换时机进行预测(在每轮迭代计算之前预测是否切换)。第 1 种方式只需运行 1 次随机森林分类器,故切换开销小,但其无法利用迭代计算过程中产生的信息进行预测,而只能通过迭代计算前的输入信息和先验知识进行预测,故在未知分布的图数据集上可能表现出较差的泛化能力。第 2 种方式需要在每一轮迭代计算之前运行 1 次分类器,且需要收集迭代计算过程中产生的特定信息,故切换开销较大,但其能够利用最新产生的迭代计算信息辅助预测,相比第 1 种方式能够更灵活地适应未知分布的图数据集。

为了充分验证混合增量计算模型的有效性,本文在 RMAT 生成图和真实图 2 类图数据集上均进行了实验。由于 RMAT 图数据集的分布较为一致,因此我们采用上述第 1 种方式预测 2 种增量计算模式的切换时机,以实现较低的切换开销。而对于真实图数据集,为了能够适应真实场景中复杂的图数据特征,我们采用上述第 2 种方式灵活地预测是否在下一轮迭代计算进行切换。我们为 RMAT 生成图数据集和真实图数据集分别训练了 2 个随机森林分类器,下面对训练过程进行详细介绍。

1)RMAT 生成图数据集分类器

我们将 BFS, SSSP, PageRank, Label Propagation 这 4 种图算法分别编号为 1, 2, 3, 4,选择 RMAT 图数据集的规模、平均度数、更新边流大小和图算法编号 4 个参数作为特征数据。特征数据可以表示为四元组:

$$\left(S, D_{\text{avg}}, |\Delta G|, A_{\text{id}}\right), \tag{14}$$

其中 S 表示图数据集的规模(顶点数量为 2^S), D_{avg} 表示图数据集的平均度数, A_{id} 表示图算法的编号。

我们将 IncRefine 和 IncRecpt 这 2 种增量计算模式分别编号为 0 和 1,为了降低预测的复杂度,我们最多进行 2 次切换。目标数据可以表示成三元组:

$$(M_{\text{id}}, I_{\text{forth}}, I_{\text{back}})。 \tag{15}$$

其中 M_{id} 表示在第 1 轮迭代计算所采用的增量计算模式, $M_{\text{id}}=0$ 表示采用 IncRefine 模式, $M_{\text{id}}=1$ 表示采用 IncRecpt 模式; I_{forth} 表示从哪一轮迭代计算开始由 M_{id} 代表的模式切换到另一种模式(即第 1 次切换); I_{back} 表示从哪一轮迭代计算开始切换回由 M_{id} 代表的模式(即第 2 次切换)。若 $I_{\text{forth}}=-1$,则不进行切换,若 $I_{\text{forth}}\neq -1$ 且 $I_{\text{back}}=-1$,则进行 1 次切换,若 $I_{\text{forth}}\neq -1$ 且 $I_{\text{back}}\neq -1$,则进行 2 次切换。

经过测试,RMAT 生成图数据集分类器的预测准确率为93%,平均预测时间开销为 7×10^{-5} s(在图算法总体执行时间中占比小于 1%)。

2)真实图数据集分类器

在迭代计算过程中,除了式(14)所采用的 4 个参数以外,我们可以另外收集每轮迭代计算的活跃顶点数量、活跃顶点的平均度数、所采用的增量计算模式种类以及图算法执行时间等作为特征数据,在下一轮迭代计算开始之前预测是否切换。收集活跃顶点的平均度数需要访问所有活跃顶点的度大小,该过程耗时较长,会较大地降低图算法的总体执行时间。因此,我们只额外增加每轮迭代计算的活跃顶点数量、所采用的增量计算模式种类以及图算法执行时间 3 个参数。假设图算法共进行了 k 轮迭代计算,特征数据可以表示为七元组:

$$\left(n, D_{\text{avg}}, |\Delta G|, A_{\text{id}}, N_{\text{f}}^i, t^i, M_{\text{id}}^i\right), i=0,1,\cdots,k-1。 \tag{16}$$

其中 N_{f}^i 和 t^i 分别表示第 i 轮迭代计算的活跃顶点数量和图算法执行时间; M_{id}^i 表示第 i 轮迭代计算所采用的增量计算模式, $M_{\text{id}}^i=0$ 表示采用了 IncRefine 模式, $M_{\text{id}}^i=1$ 表示采用了 IncRecpt 模式。需要注意的是,在第 1 轮迭代计算之前需要对第 1 轮迭代计算采用何种增量计算模式进行预测(即 $i=0$ 时),我们规定 $N_{\text{f}}^0=0$, $t^0=0$, $M_{\text{id}}^0=-1$。

对于第 1 轮迭代计算之前的预测,我们设置目标数据为 2 种增量计算模式的图算法总体执行时间。而对于其他轮迭代计算之前的预测,我们设置目标数据为 2 种增量计算模式在每轮迭代计算的图算法执行时间。目标数据可以表示为二元组:

$$\left(t_{i+1}^{\text{refine}}, t_{i+1}^{\text{recpt}}\right), i=0,1,\cdots,k-1, \tag{17}$$

其中当 $i=0$ 时, t_1^{refine} 和 t_1^{recpt} 分别表示 IncRefine 和 IncRecpt 模式的图算法总体执行时间,而 $i>0$ 时, t_{i+1}^{refine} 和 t_{i+1}^{recpt} 分别表示 IncRefine 和 IncRecpt 模式在第 $i+1$ 轮迭代计算的图算法执行时间。

经过测试,真实图数据集分类器的预测准确率为89%,每轮迭代计算的平均预测时间开销 $t_{\text{cost}}=2\times10^{-4}$ s。由于图算法需要进行多次迭代计算,故累积的总体预测时间开销为毫秒级别,在图算法总体执行时间中的占比最高约为10%。

在迭代计算过程中,当其中一种增量计算模式的性能总是比另一种增量计算模式的性能更优时,通过真实图数据集分类器进行预测不仅不会带来性能增益,反而会额外增加预测时间开销。因此,我们设定阈值 α(默认为 0.5),对于第 1 轮迭代计算之前的预测,当

$$\frac{\left|t_1^{\text{refine}}-t_1^{\text{recpt}}\right|}{\min(t_1^{\text{refine}},t_1^{\text{recpt}})}>\alpha \tag{18}$$

时，在整个迭代计算过程中以性能更优的增量计算模式执行图算法，而不使用分类器进行预测。另外，当2种增量计算模式在第$i+1$轮迭代计算（$i>0$）的图算法执行时间差不大于预测时间开销时，切换带来的收益会小于其带来的开销。因此，我们设定迭代计算过程中的切换条件为

$$\left|t_{i+1}^{\text{refine}}-t_{i+1}^{\text{recpt}}\right|>t_{\text{cost}},\ i=1,2,\cdots,k-1。 \tag{19}$$

当式(19)成立时，才从性能较差的增量计算模式切换到性能较优的增量计算模式。

5 实验评估

本节首先介绍实验设置，然后分别在RMAT生成图数据集和真实图数据集上对IncHybrid模式的性能进行评估。

5.1 实验设置

1)实验平台

本文采用带有Intel® Xeon® Platinum 8269CY处理器(26核52线程，主频2.50 GHz)和256 GB DDR4内存的计算机，该机器运行64位的CentOS 7操作系统，编译器采用了支持CilkPlus[37]并行编程框架的GCC 5.5.0(使用-O3优化)。

2)工作负载

本文采用了PageRank[1], Label Propagation[34], BFS[15], SSSP[16]这4种典型的图算法。我们设置PageRank算法的阻尼系数$d=0.85$，设置PageRank和Label Propagation算法的更新阈值$\epsilon=10^{-5}$。此外，PageRank和Label Propagation算法的迭代计算轮数上限被设置为20，而BFS和SSSP算法则迭代计算到收敛为止。

3)图数据集

本文采用RMAT[35]生成图数据集和真实图数据集。RMAT数据集是Graph500[38]榜单采用的官方数据集，其模拟了真实图数据的特征。我们设置参数$a=0.57$, $b=c=0.19$，生成了9个不同顶点规模和平均度数的RMAT有向图数据集。对于真实图数据集，采用了源于斯坦福大学SNAP项目的5个图数据集，如表1所示，com-LiveJournal (LJ)[39], com-Orkut (OR)[39], soc-Pokec (PK)[40]为社交网络图数据集，而roadNet-CA (CA)[41]和roadNet-TX (TX)[41]为道路网络图数据集。我们参考文献[6−7]的工作对流式图进行模拟，随机抽取图数据集50%的边构造基础图，然后从基础图中随机选择删除的边，而从剩下50%的边中随机选择插入的边，删除的边和插入的边构成更新边流ΔG，本文固定更新边流中删除边和插入边的比例为1∶1。

Table 1 Fourteen Graph Datasets Used in the Experiment

表1 实验所采用的14个图数据集

数据集	顶点数量	边数量
RMAT_S_16（共6个，$S\in\{20,21,22,23,24,25\}$）	2^S	$2^S\times16$
RMAT_24_D_{avg}（共3个，$D_{\text{avg}}\in\{10,13,19\}$）	2^{24}	$2^{24}\times D_{\text{avg}}$
LJ	3 997 962	34 681 189
OR	3 072 441	117 185 083
PK	1 632 803	30 622 564
CA	1 965 206	2 766 607
TX	1 379 917	1 921 660

注：S表示图数据集的规模（顶点数量为2^S），D_{avg}表示图数据集的平均度数.

5.2 RMAT生成图数据集上的实验结果

1)性能评估

图3给出了当$|\Delta G|=1\,000$时，4种图算法在RMAT_S_16($S\in\{20,21,22,23,24,25\}$)这6个数据集上IncHybrid, IncRefine, IncRecpt模式下相比Tradition模式的加速比。BFS, SSSP, PageRank, Label Propagation这4种图算法在IncHybrid模式下相比Tradition模式平均实现了2.56、2.40、1.62和1.30的加速比。从图3(a)(b)中可以看出，对于BFS和SSSP算法，IncRefine模

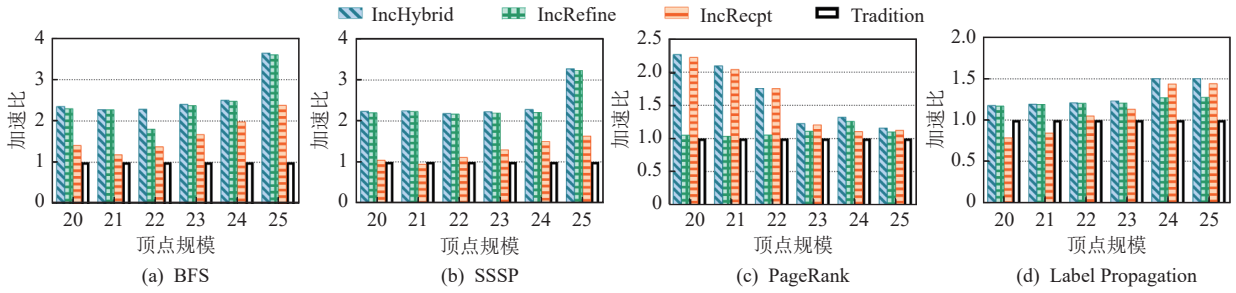


Fig. 3 Performance comparison of four computation modes on RMAT dataset with varied scales of vertex

图3 4种计算模式在不同顶点规模的RMAT数据集上的性能比较

式相比 IncRecpt 模式具有更优的整体性能，而它们相比 Tradition 模式都具有一定的提升，通过切换，IncHybrid 模式成功达到了与 IncRefine 模式接近的性能。从图 3(c)(d)中可以看出，PageRank 和 Label Propagation 算法在 IncRefine 模式和 IncRecpt 模式下不同顶点规模的数据集上呈现的性能优劣不同。例如，当顶点数据规模不超过 23 时，Label Propagation 算法在 IncRefine 模式下具有更优的性能，但当顶点规模为 24 和 25 时 IncRecpt 模式具有更优的性能。从中可以看出，在 2 种增量计算模式中 IncHybrid 模式成功达到了最佳的性能，这表明切换机制能够有效地采用性能更优的增量计算模式。

图 4 给出了当$|\Delta G|=1\,000$时，4 种图算法在 RMAT_24_D_{avg}($D_{avg}\in\{10,13,16,19\}$)这 4 个数据集上分别采用 IncHybrid, IncRefine, IncRecpt, Tradition 模式的执行时间。BFS, SSSP, PageRank, Label Propagation 算法在 IncHybrid 模式下相比 Tradition 模式实现了平均 2.43、2.27、1.22和1.54的加速比。从图 4 中可以看出，当平均度数变化时，IncHybrid 模式的性能与 IncRefine 和 IncRecpt 这 2 种模式中性能更优的那个的性能接近，这表明 IncHybrid 模式能够有效适应不同平均度数的数据集。图 5 给出了在 RMAT_25_16数据集上，当更新边流大小$|\Delta G|$变化时，BFS 和 PageRank 算法在 IncHybrid, IncRefine, IncRecpt 模式下的执行时间。

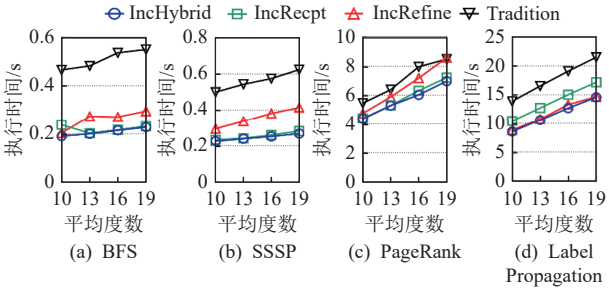


Fig. 4 Performance comparison of four computation modes on RMAT dataset with varied average degrees

图 4 4 种计算模式在不同平均度数的 RMAT 数据集上的性能比较

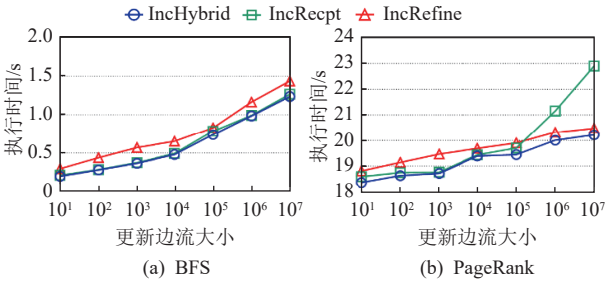


Fig. 5 Performance comparison of four computation modes on RMAT_25_16 when $|\Delta G|$ varies

图 5 当$|\Delta G|$变化时 4 种计算模式在 RMAT_25_16 上的性能比较

从图 5 中可以看出，当$|\Delta G|$增大时，2 种算法在 3 种增量计算模式下的执行时间均有所增加，这是因为受更新边流影响需要参与计算的顶点数量变多了，而 IncHybrid 模式仍然与 IncRefine 和 IncRecpt 这 2 种模式中性能更优的那个的性能接近，这表明其能够有效地适应不同的更新边流大小。

2)切换效果分析

图 6 给出了当$|\Delta G|=10$时，4 种图算法在 RMAT_25_16 数据集上分别采用 IncHybrid, IncRefine, IncRecpt 模式的每轮迭代计算时间，实验结果表明 IncHybrid 模式能够有效进行切换并取得最优性能。

对于 BFS 算法，随机森林分类器的预测值为(1,2,6)，IncHybrid 模式在第 1 轮迭代计算采用 IncRecpt 模式，然后在第 2 轮迭代计算切换到 IncRefine 模式，最后在第 6 轮迭代计算切换回 IncRecpt 模式。从图 6(a)中可以看出，IncHybrid 模式在每轮迭代计算中取得了较优的性能。最终 BFS 算法在 IncHybrid 模式下取得了0.20 s 的执行时间，而在 IncRefine 和 IncRecpt 模式分别取得了0.21 s 和0.30 s 的执行时间。

对于 SSSP 算法，随机森林分类器的预测值为(1,2,7)，IncHybrid 模式在第 1 轮迭代计算采用 IncRecpt 模式，随后在第 2 轮迭代计算切换到 IncRefine 模式，最后在第 7 轮迭代计算切换回 IncRecpt 模式。从图 6(b)中可以看出，IncHybrid 模式成功切换到指定的模式

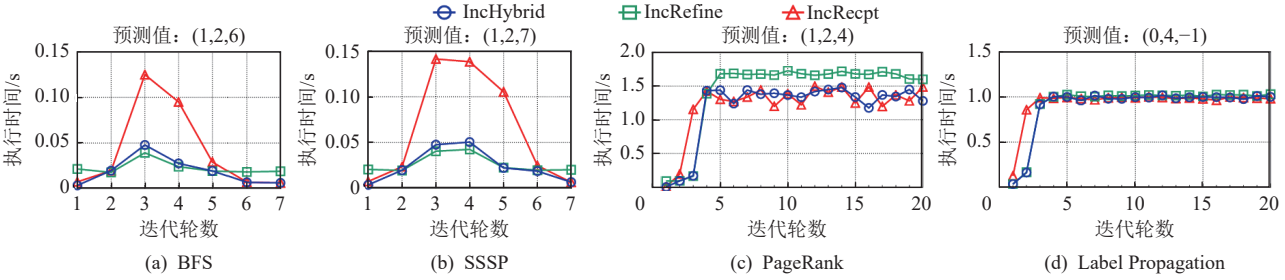


Fig. 6 Switching effect analysis of IncHybrid mode

图 6 IncHybrid 模式的切换效果分析

并获得了最优的整体性能。SSSP 算法在 IncHybrid 模式下的执行时间为0.24 s，而在 IncRefine 和 IncRecpt 模式下的执行时间分别为0.26 s 和0.46 s。

对于 PageRank 算法，随机森林分类器的预测值为(1,2,4)，IncHybrid 模式在第 1 轮迭代计算采用了 IncRecpt 模式，接着在第 2 轮迭代计算切换到 IncRefine 模式，之后在第 4 轮迭代计算切换回 IncRecpt 模式，如图 6(c)所示。PageRank 算法在 IncHybrid 模式下取得了18.36 s 的执行时间，而在 IncRefine 和 IncRecpt 模式下分别取得了18.59 s 和18.81 s 的执行时间。

对于 Label Propagation 算法，随机森林分类器的预测值为(0,4,−1)，IncHybrid 模式在第 1 轮迭代计算采用 IncRefine 模式，然后在第 4 轮迭代计算切换到 IncRecpt 模式，并在之后的迭代计算中继续采用 IncRecpt 模式，如图 6(d)所示。最终 Label Propagation 算法在 IncHybrid 模式下的执行时间为24.05 s，而在 IncRefine 和 IncRecpt 模式下的执行时间分别为28.87 s 和24.43 s。

3)与其他系统的比较

我们将 IncHybrid 模式和先进的流式图处理系统 DZIG[19] 进行了比较，DZIG 系统采用了基于修正的增量计算模式，并同样使用基于 BSP 的以顶点为中心的迭代计算模型。由于 DZIG 系统的开源代码中未提供 BFS 和 SSSP 算法，所以我们在 DZIG 系统中实现了该 2 种算法。

表 2 给出了当更新边流大小$|\Delta G|=1\,000$时，在 RMAT_S_16($S\in\{21,22,23,24\}$)这 4 个数据集上，PageRank, Label Propagation, BFS, SSSP 这 4 种图算法在 IncHybrid 模式下以及采用 DZIG 系统的执行时间。总体上，IncHybrid 模式相比 DZIG 系统实现了平均1.25的加速比。从表 2 中可以看出，IncHybrid 模式在大多数情况下的性能优于 DZIG 系统，其主要原因在于 IncHybrid 模式在迭代计算过程中会准确切换到 IncRefine 和 IncRecpt 模式中性能更优的模式。其中在 RMAT_21_16 数据集上，PageRank 算法在 IncHybrid 模式下相比 DZIG 系统达到了 1.92 的加速比，这是因为 IncHybrid 模式在第 1 轮迭代计算中采用了 IncRefine 模式，而从第 2 轮迭代计算开始切换到了性能更优的 IncRecpt 模式。在某些情况下 IncHybrid 模式与 DZIG 系统的性能接近(例如 Label Propagation 算法在 RMAT_21_16, RMAT_22_16, RMAT_23_16 这 3 个数据集上)，这是因为 IncHybrid 模式从第 1 轮迭代计算开始只采用了 IncRefine 模式而没有进行切换。

5.3 真实图数据集上的实验结果

图 7 给出了当$|\Delta G|=1\,000$时，BFS 和 SSSP 算法在 5 个真实图数据集上分别采用 IncHybrid, IncRefine, IncRecpt 模式相比 Tradition 模式的加速比(我们省略了 PageRank 和 Label Propagation 算法，因为它们表现出与图 7 相似的结果)。

对于 BFS 算法，IncHybrid, IncRefine, IncRecpt 模式相比 Tradition 模式的加速比分别为4.10、2.65和3.55。对于 SSSP 算法，IncHybrid, IncRefine, IncRecpt 模式相比 Tradition 模式的加速比分别为3.46、2.67和3.45。我们发现当 IncRefine 和 IncRecpt 模式的性能较为接近时，IncHybrid 模式能够有效地在迭代计算过程中选择并切换到性能更优的模式(例如图 7(a)的 OR 数据集)，从而获得更短的算法执行时间。从图 7 中可以看出，BFS 算法在 LJ 数据集上以及 SSSP 算法在 LJ 和 TX 数据集上，IncRecpt 模式的性能全过程优于 IncRefine 模式，通过在迭代计算之前比较预测出的 2 种增量计算模式的图算法总体执行时间(计算见式(18))，IncHybrid 模式避免了不必要的预测时间开销，从而达到与 IncRecpt 模式相近的性能。

6 结　论

本文将增量计算归纳为基于修正的增量计算模式和基于重计算的增量计算模式，在对二者进行理

Table 2 Execution Time and Speedup of IncHybrid Mode and DZIG System

表 2 IncHybrid 模式与 DZIG 系统的执行时间和加速比

算法	执行时间/s								IncHybrid 模式相比 DZIG 系统的加速比			
	DZIG 系统				IncHybrid 模式（本文）							
	21	22	23	24	21	22	23	24	21	22	23	24
PageRank	1.014	1.606	2.704	6.538	0.527	1.018	2.568	6.302	1.92	1.58	1.05	1.04
Label Propagation	2.330	4.117	7.444	14.995	2.353	4.162	7.361	12.674	0.99	0.99	1.01	1.18
BFS	0.032	0.068	0.150	0.275	0.029	0.049	0.095	0.217	1.10	1.39	1.58	1.27
SSSP	0.037	0.071	0.153	0.329	0.033	0.057	0.115	0.252	1.12	1.25	1.33	1.31

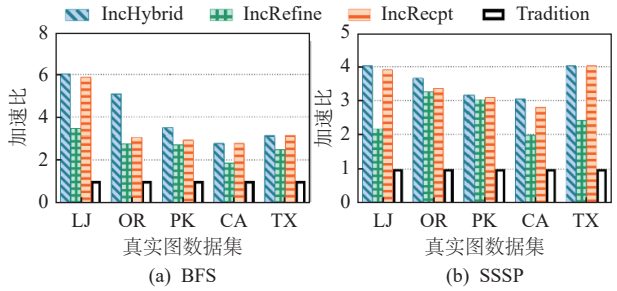


Fig. 7 Performance comparison of four computation modes on the real-world datasets

图 7 4 种计算模式在真实图数据集上的性能比较

论和实验分析的基础上,提出了混合增量计算模式,实现了在 2 种增量计算模式之间正确且有效地切换。本文提出的基于混合增量计算模式的流式图并行处理,在不同规模数据集、不同图算法以及不同更新边流大小下有效地实现了切换,相比先进的流式图处理系统 DZIG 实现了平均1.25倍的性能提升。未来我们将探索采用其他预测模型进行优化,以更好地提升流式图处理的性能。

作者贡献声明:申玥负责方案整体设计、代码和论文的撰写;张婕、张园、曹华伟负责提出部分实验思路并修改论文;安学军、叶笑春提出指导意见。

参 考 文 献

[1] Berkhin P. A survey on PageRank computing[J]. Internet Mathematics, 2005, 2(1): 73−120

[2] Afarin M, Gao Chao, Rahman S, et al. CommonGraph: Graph analytics on evolving data[C]//Proc of the 28th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2023: 133−145

[3] Zhang Yu, Liang Yuxuan, Zhao Jin, et al. Egraph: Efficient concurrent GPU-based dynamic graph processing[J]. IEEE Transactions on Knowledge and Data Engineering, 2022, 35(6): 5823−5836

[4] Han Wentao, Miao Youshan, Li Kaiwei, et al. Chronos: A graph engine for temporal graph analysis[C]//Proc of the 9th European Conf on Computer Systems. New York: ACM, 2014: 1−14

[5] Miao Youshan, Han Wentao, Li Kaiwei, et al. Immortalgraph: A system for storage and analysis of temporal graphs[J]. ACM Transactions on Storage, 2015, 11(3): 1−34

[6] Vora K, Gupta R, Xu Guoqing. KickStarter: Fast and accurate computations on streaming graphs via trimmed approximations[C]//Proc of the 22nd Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2017: 237−251

[7] Mariappan M, Vora K. GraphBolt: Dependency-driven synchronous processing of streaming Graphs[C]//Proc of the 14th EuroSys Conf 2019. New York: ACM, 2019: 1−16

[8] Iyer A P, Pu Qifan, Patel K, et al. TEGRA: Efficient ad-hoc analytics on evolving graphs[C]//Proc of the 18th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2021: 337−355

[9] Dhankhad S, Mohammed E, Far B. Supervised machine learning algorithms for credit card fraudulent transaction detection: A comparative study[C]//Proc of the 19th IEEE Int Conf on Information Reuse and Integration (IRI). Piscataway, NJ: IEEE, 2018: 122−125

[10] Gupta P, Satuluri V, Grewal A, et al. Real-time Twitter recommendation: Online motif detection in large dynamic graphs[J]. Proceedings of the VLDB Endowment, 2014, 7(13): 1379−1380

[11] Kwak H, Lee C, Park H, et al. What is Twitter, a social network or a news media?[C]//Proc of the 19th Int Conf on World Wide Web. New York: ACM, 2010: 591−600

[12] Besta M, Fischer M, Kalavri V, et al. Practice of streaming processing of dynamic graphs: Concepts, models, and systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 34(6): 1860−1876

[13] Zhang Yuan, Cao Huawei, Zhang Jie, et al. Survey on key technologies of graph processing systems based on multi-core CPU and GPU platforms[J]. Journal of Computer Research and Development, 2024, 61(6): 1401−1428 (in Chinese)
(张园, 曹华伟, 张婕, 等. 面向多核 CPU 与 GPU 平台的图处理系统关键技术综述[J]. 计算机研究与发展, 2024, 61(6): 1401−1428)

[14] Fan Wenfei, Tian Chao. Incremental graph computations: Doable and undoable[J]. ACM Transactions on Database Systems, 2022, 47(2): 1−44

[15] Beamer S, Asanović K, Patterson D. Direction optimizing breadth first search[J]. Scientific Programming, 2013, 21(3/4): 137−148

[16] Chakaravarthy V T, Checconi F, Murali P, et al. Scalable single source shortest path algorithms for massively parallel systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 28(7): 2031−2045

[17] Feng Guanyu, Ma Zixuan, Li Daixuan, et al. RisGraph: A real-time streaming system for evolving graphs to support sub-millisecond per update analysis at millions Ops/s[C]//Proc of the 48th Int Conf on Management of Data. New York: ACM, 2021: 513−527

[18] Jiang Xiaolin, Xu Chengshuo, Yin Xizhe, et al. Tripoline: Generalized incremental graph processing via graph triangle inequality[C]//Proc of the 16th European Conf on Computer Systems. New York: ACM, 2021: 17−32

[19] Mariappan M, Che J, Vora K. DZIG: Sparsity-aware incremental processing of streaming graphs[C]//Proc of the 16th European Conf on Computer Systems. New York: ACM, 2021: 83−98

[20] Gong Shufeng, Tian Chao, Yin Qiang, et al. Ingress: An automated incremental graph processing system[J]. The VLDB Journal, 2024, 33(3): 781−806

[21] Murray D G, McSherry F, Isaacs R, et al. Naiad: A timely dataflow system[C]//Proc of the 24th ACM Symp on Operating Systems Principles. New York: ACM, 2013: 439−455

[22] Carbone P, Katsifodimos A, Ewen S, et al. Apache Flink: Stream and batch processing in a single engine[J]. The Bulletin of the Technical Committee on Data Engineering, 2015, 38(4): 28−38

[23] Shi Xiaogang, Cui Bin, Shao Yingxia, et al. Tornado: A system for real-time iterative analysis over evolving data[C]//Proc of the 43rd Int Conf on Management of Data. New York: ACM, 2016: 417−430

[24] Toshniwal A, Taneja S, Shukla A, et al. Storm@ Twitter[C]//Proc of the 41st Int Conf on Management of Data. New York: ACM, 2014:

147−156
[25] Chen Dan, Gui Chuangyi, Zhang Yi, et al. GraphFly: Efficient asynchronous streaming graphs processing via dependency-flow[C]//Proc of the 35th Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2022: 1−14
[26] Jiang Zihan, Mao Fubing, Guo Yapu, et al. ACGraph: Accelerating streaming graph processing via dependence hierarchy[C]//Proc of the 60th ACM/IEEE Design Automation Conf (DAC). Piscataway, NJ: IEEE, 2023: 1−6
[27] Valiant L G. A bridging model for parallel computation[J]. Communications of the ACM, 1990, 33(8): 103−111
[28] Malewicz G, Austern M H, Bik A J C, et al. Pregel: A system for large-scale graph processing[C]//Proc of the 37th Int Conf on Management of Data. New York: ACM, 2010: 135−146
[29] Shun Julian, Blelloch G E. Ligra: A lightweight graph processing framework for shared memory[C]//Proc of the 18th ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2013: 135−146
[30] Kyrola A, Blelloch G, Guestrin C. GraphChi: Large-scale graph computation on just a PC[C]//Proc of the 10th USENIX Conf on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2012: 31−46
[31] Zhu Xiaowei, Han Wentao, Chen Wenguang. GridGraph: Large scale graph processing on a single machine using 2-level hierarchical partitioning[C]//Proc of the 21st USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2015: 375−386
[32] Low Y, Gonzalez J, Kyrola A, et al. GraphLab: A new framework for parallel machine learning[C]//Proc of the 26th Conf on Uncertainty in Artificial Intelligence. Arlington, Virginia: AUAI, 2010: 340−349
[33] Ediger D, McColl R, Riedy J, et al. Stinger: High performance data structure for streaming graphs[C]//Proc of the 16th IEEE Conf on High Performance Extreme Computing. Piscataway, NJ: IEEE, 2012: 1−5
[34] Iscen A, Tolias G, Avrithis Y, et al. Label propagation for deep semi-supervised learning[C]//Proc of the 32nd IEEE/CVF Conf on Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2019: 1−10
[35] Chakrabarti D, Zhan Yiping, Faloutsos C. RMAT: A recursive model for graph mining[C]//Proc of the 4th SIAM Int Conf on Data Mining. Philadelphia, PA: SIAM, 2004: 442−446
[36] Breiman L. Random forests[J]. Machine Learning, 2001, 45: 5−32
[37] Leiserson C E. The Cilk++ concurrency platform[C]//Proc of the 46th Annual Design Automation Conf. Piscataway, NJ: IEEE, 2009: 522−527
[38] Bader D A, Berry J, Kahan S, et al. Graph500[EB/OL]. 2010[2025-05-05]. http://www.graph500.org
[39] Yang Jaewon, Leskovec J. Defining and evaluating network communities based on ground-truth[C]//Proc of the 18th ACM SIGKDD Workshop on Mining Data Semantics. New York: ACM, 2012: 1−8
[40] Takac L, Zabovsky M. Data analysis in public social networks[C/OL]//Proc of the Int Scientific Conf and Int Workshop Present Day Trends of Innovations. 2012[2025-05-05]. https://snap.stanford.edu/data/soc-pokec.pdf
[41] Leskovec J, Lang K J, Dasgupta A, et al. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters[J]. Internet Mathematics, 2009, 6(1): 29−123



Shen Yue, born in 1999. PhD candidate. Member of CCF. Her main research interests include parallel computing and dynamic graph processing.
申　玥, 1999 年生。博士研究生。CCF 会员。主要研究方向为并行计算、动态图处理。



Zhang Jie, born in 1999. Master, assistant engineer. Her main research interests include high-performance computing and graph algorithm optimization.
张　婕, 1999 年生。硕士，助理工程师。主要研究方向为高性能计算、图算法优化。



Zhang Yuan, born in 1990. PhD, engineer. Member of CCF. Her main research interests include high-performance computing and graph algorithm optimization.
张　园, 1990 年生。博士，工程师。CCF 会员。主要研究方向为高性能计算、图算法优化。



Cao Huawei, born in 1989. PhD, associate professor, master supervisor. Member of CCF. His main research interests include parallel computing and high throughput computing architecture.
曹华伟, 1989 年生。博士，副研究员，硕士生导师。CCF 会员。主要研究方向为并行计算、高通量计算架构。



An Xuejun, born in 1966. PhD, senior engineer, PhD supervisor. Member of CCF. His main research interests include computer system architecture and high-performance interconnection networks.
安学军, 1966 年生。博士，高级工程师，博士生导师。CCF 会员。主要研究方向为计算机系统结构、高性能互连网络。



Ye Xiaochun, born in 1981. PhD, professor, PhD supervisor. Member of CCF. His main research interests include algorithm paralleling and optimizing software simulation, and architecture for high throughput computer.
叶笑春, 1981 年生。博士，研究员，博士生导师。CCF 会员。主要研究方向为算法并行与优化软件仿真、高通量计算机体系结构。