高技术通讯 2022 年 第32 卷 第8 期:825-835

doi:10.3772/j.issn.1002-0470.2022.08.005

基于块坐标下降法的外存异步图计算系统①

赵 程② 张志斌③ 郭嘉丰 刘丁玮

(中国科学院计算技术研究所网络数据科学与技术重点实验室 北京 100190)

(中国科学院大学 北京 100049)

摘 要 现有外存图计算系统中,外存 I/O 带宽不足成为性能瓶颈。使用整体并行模型将导致冗余计算,使用异步并行模型则将引入额外的优先级计算开销和负载不均衡。本文提出了基于块坐标下降法(BCD)的外存异步图计算系统(BCDG),设计了一次选择多轮优先的调度策略,降低了优先级计算的平均开销;设计了基于优先级的块预取策略,解决了优先选择会破坏顺序执行流水线的问题;设计了计算调度分离的划分策略,实现了均衡地按边计算和按点调度。实验结果表明,相较于目前最先进的外存图计算系统 GridGraph 和 Lumos,所提系统平均性能分别提升 10.30 倍与 8.72 倍。整体计算过程中,中央处理器(CPU)等待外存 I/O 的时间仅占 10% ~30%。

关键词 图计算系统; 外存; 异步并行计算; 块坐标下降(BCD); 图划分

0 引 言

图数据普遍存在于各个领域,如社交网络、网页链接、金融网络、生物网络等,其中大图计算任务吸引了大量的关注,构建高效易用的图计算系统迫在眉睫。近些年的研究设计并实现了基于各类计算资源的高性能图计算系统[1-9],其中,尽可能地利用外存扩展处理数据的规模,并且探索更高效的外存计算模式成为图计算系统扩展能力研究的关键。

大部分基于外存的系统[2,7-8]在执行图计算的任务时采用整体同步并行计算(bulk synchronous parallel, BSP)模型,在图计算的每轮迭代中都需要从外存加载全量图数据进行计算。外存图计算系统使用相对简单的划分算法划分图数据,并据此安排流水线模式,通过计算时间掩盖加载开销。然而,它们依据划分信息顺序执行,每轮计算中的数据块最多处理一次,从而造成数据的重复加载与计算,浪费了外存带宽。异步计算模式的外存图计算系统通过有选择的加载数据块并对其进行多次计算来减少外存加载的冗余,如 Clip[5]、Lumos[3]等。然而,Lumos 设置了每个数据块计算的硬上限,而 Clip 缺少对内存异步执行程度的控制,它们只是一定程度上缓解了冗余加载的问题。

使用基于优先选择的异步图计算仍然存在诸多挑战:第一,计算优先级需要综合全图点信息,需要额外开销;第二,BSP 模型系统中的加载计算流水线设计得益于其顺序执行策略,而优先选择的策略会破坏这种顺序执行流程;第三,由于图数据的偏斜性[10],基于点集的优先级选择策略要求每个数据块包含相同的点数,这会造成图计算的负载不均衡。

本文提出了一个基于块坐标下降法(block coordinate descent, BCD)的外存异步图计算系统(BCD based asynchronous out-of-core graph computing system, BCDG)。首先将图算法转化为最优化问

① 中国科学院战略性先导科技专项(XDA19020400),中国科学院青年创新促进会(20144310),联想-中科院联合实验室青年科学家项目(E051350),重庆市基础科学与前沿技术研究专项(cstc2017jcjyBX0059)资助项目。

② 男,1994 年生,博士生;研究方向:图计算系统;联系人,E-mail: zhaocheng@ict.ac.cn。

③ 通信作者,E-mail: zhangzhibin@ict.ac.cn。

(收稿日期:2021-09-17)

题,然后使用块坐标下降法作为块优先级算法,每次计算选择对算法收敛贡献最大的块,以减少冗余计算。利用 BCD 的优先级序列能够预测后续若干轮选择的特性,设计了一次选择多轮优先的调度策略,从而降低了优先级计算的平均开销。基于此特性,设计了后续计算的预取机制和对应的计算加载流水线,填补了中央处理器(central processing unit, CPU)计算优先级时加载线程的空闲。考虑到负载均衡问题,设计了计算调度分离的划分策略,根据边将图平均分为一个个分片,并将一个分片视为 BCDG 中的最小计算调度单元。调度时将分片按点集合拼成 BCD 选中的数据块,计算时按边对这些分片并发计算。使用 BCDG 的系统框架实现了图计算中的经典问题 PageRank 和弱连通分量,并进行了充分的实验验证。

1 相关工作

目前,外存图计算系统设计外存友好的数据结构以及对应的划分策略,并且据此设计对应的执行引擎。这些设计往往采用流式地加载、计算划分后的数据库的方式,以求最大化数据处理的局部性。下面简要讨论近年来一些相关工作。

同步外存图计算系统。Graphene[2] 使用 I/O 请求为中心的图处理模型,通过将高层数据访问转化为细粒度的 I/O 请求来简化 I/O 管理。DynamicShards[11] 通过动态分区,从分区中移除不必要的边,以减少磁盘 I/O。GraphOne[12] 使用了邻接表和压缩系数矩阵两套结构存储图数据以实现部分的动态图数据处理的负载能力。

异步外存图计算系统。Wonderland[13] 抽取有效的图摘要以捕捉特定的图属性,在图摘要的引导下执行处理,推断出更好的优先处理顺序和更快的图信息传播。虽然这种基于图摘要的技术很有效,但其应用范围仅限于基于路径的单调图算法,此外算法的适用性仍未确定。AsyncStripe[14] 使用非对称分区和基于条带的自适应访问策略来处理异步算法。由于这些工作没有提供同步保证,它们仅适用于基于路径的异步算法。文献[15]针对 libaio 引擎设计了自适应的 I/O 和计算调度机制并利用批(batch)机制平衡了异步计算的负载。

外存计算之外的工作。Gemini[1] 和 SympleGraph[6] 提供了分布式图计算的同步处理模型。文献[16]探讨了分布式图计算时的容错计算问题。Teseo[17] 关注动态图的存储并提供同步处理模型。Kaleido[18] 针对图挖掘类任务以子图为中心的模型计算提供了同步处理模型。

2 背景与动机

本节首先讨论最先进的外存图计算系统,然后介绍整体同步并行、异步并行模式在图计算中的应用,随后分析在外存图计算系统中应用异步并行的动机与挑战。

2.1 基于外存的图计算系统

基于外存的单机图计算系统对于图划分问题采用不需要复杂计算的线性划分策略,然后以流式的方式按照划分后的数据块计算图数据,以此最大化利用图数据的顺序局部性。

同步并行计算。GraphChi[7] 在点集上进行图划分,并保证每个分块包含的边数尽可能相同。GraphChi 为每个图划分在内存中维护内存缓冲区,当缓冲区耗尽时从外存中加载划分数据,并且使用协程来实现重叠计算与加载的时间。GridGraph[8] 将点集划分为若干子集,将边集划分为基于点划分子集的二维网络;然后构建基于二维数据块的流式执行,根据不同的算法优先选择按行或按列执行。加载任务被拆分成小块,以队列的形式维护到协程中,从而实现加载与计算的重叠。

缺陷:上述外存图计算系统根据图数据的划分信息顺序执行,其数据划分的计算顺序是在计算前确定的,并且每轮计算中每个加载的块最多处理一次。这类预先确定执行顺序和最多处理一次的限制浪费了宝贵的外存带宽,造成了重复的数据加载与计算。

异步并行计算。上述系统除了支持同步并行计算模型,还支持简单的异步执行,在图算法迭代过程中允许更新函数使用最新的中间结果,然而每个加

载的块在每轮迭代中最多仍然只能处理一次。Lumos[3]在 GridGraph 的基础上,使用无序的异步执行在多轮计算中共用计算结果,以减少外存加载。Clip[5]针对此设计了同步与异步计算的折中,其在块之间进行同步处理以确保磁盘的顺序 I/O,在每个块内实现异步处理。

缺陷:Lumos 一定程度上缓解了每个加载块在每轮计算中最多处理一次的问题,但缓解程度受限于其设定的硬上限。Clip 缺少对于内存异步执行程度的控制,整体设计通过块内的异步计算的 CPU 资源换取外存 I/O,仍然存在冗余计算与加载。Clip 仅提供了异步算法的支持,如广度优先搜索、最短路径等,对于传统的同步算法如 PageRank 支持能力有限。

2.2　动机与挑战

近年来,有些在分布式场景和异构计算场景的图计算系统使用块选择策略在图算法迭代过程中优先选择“重要”的数据块进行计算,以此加速图算法收敛。分布式图计算系统 PrIter[19]和 Maiter[20]的优先选择策略是利用基于差值的计算方法选择并更新每轮图算法迭代中差值最大的数据块。基于 CPU-FPGA 的异构计算图计算系统 GraphABCD[21]将图算法转化为最优化问题,并使用块坐标下降方法,每轮选择对目标函数下降最大的块进行计算,以此最大程度地减少冗余计算。这些系统在多计算资源(分布式 CPU、CPU-FPGA)环境中验证了对图划分进行优先选择策略能够减少冗余计算、提升收敛效率。在外存图计算系统中,优先选择策略有助于计算过程中总是计算对于算法收敛帮助最大的数据块,从而最大程度地提升算法收敛效率并减少冗余的数据加载。

在外存图计算系统中应用异步并行的挑战有:第一,由于计算数据块被选择的优先级带来的每轮额外选择时间开销不容忽略。数据块的优先级与块内容以及计算中间结果息息相关[19],需要综合块的数量、块中包含的点信息以及点对应的中间结果进行计算;第二,基于优先选择设计外存图计算框架还需要进一步设计加载、计算的流水线。由于优先选择策略的计算模式为选择-计算,每轮计算依赖于上一轮的计算结果,串行的执行选择、加载、计算操作将会浪费大量 CPU 时间。第三,基于点集的数据块的优先级选择会造成图计算的负载不均衡。考虑到优先选择的计算量,不论 PrIter 还是基于 BCD 的优先选择策略都是基于均分点集进行计算、选择的,图数据的偏斜性平均分割的点集极易造成负载的不均衡。

3　系统架构

为了解决上述问题,本文提出了基于最优化方法的外存图计算系统 BCDG。本节将介绍系统架构以及 BCDG 的工作流程。

如图 1 所示,本系统由图存储单元、选择单元、计算单元和加载单元组成。对于输入图,图存储单元分别将按源点和目的点排序的图数据分别存储为压缩稀疏行(compressed sparse row,CSR)和压缩稀

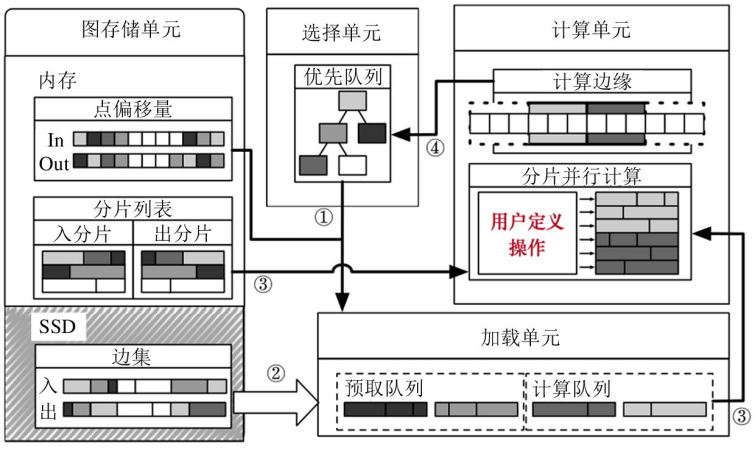


图 1　系统架构

疏列(compressed sparse column,CSC)结构,其中顶点偏移量数组存储在内存中,边集列表存储在固态硬盘(solid state disk, SSD)中。该单元在逻辑上将边集列表维护为数个分片(无数据拷贝),其中每个分片指示等量的边数据。选择单元使用优先选择策略中定义的优先级为块维护一个优先级队列。该单元计算出前 τ 个运算必要的块和需要预取的块。加载单元作为守护进程维护一个加载队列和一个预取队列,根据选择单元从 SSD 上的图存储单元加载运算必要的边数据和预取的边数据。计算单元维护图算法的计算边缘(frontier),其中包含选中的前 τ 个块和相应的顶点。该单元为在使用 push/pull 模式[4]的定制图算法提供接口。

在 BCDG 中运行迭代图算法时,首先将顶点集分为 β 个块,每个块在内存中以若干分片的形式维护,每个分片中包含相等数量的边。每次迭代中,选择单元选择优先级最高的 2τ 个块。其步骤如下:(1)选择单元告知加载单元要加载的 τ 个块和预取的 τ 个块,加载单元从内存中的图存储单元获取这些块的元数据;(2)加载单元作为协程从 SSD 中加载相应的边数据;(3)计算单元从加载单元读取边数据,并从图存储单元获取相应的数据分片信息,然后在块上并行运行用户定义的 push/pull 操作;(4)一轮迭代结束时计算单元重新计算每个块的优先级,并通知选择单元更新块的优先级队列。其中,只有步骤(2)包含外存的读操作,而其他的都是原地计算。

4 优先选择策略

将图计算问题转化为最优化问题并使用最优化问题的解决方法是目前最直观且有效的方案。沿用 GraphABCD[21]中的思路,使用块坐标下降方法解决图计算的最优化问题。本节首先介绍将图计算任务转化为最优化问题的方法,再探讨基于最优化选择的块的重用距离,最后介绍根据此设计 BCDG 的优先选择策略。

4.1 最优化图计算任务

最优化问题可以被描述为找到能够使得目标函数 $F(\boldsymbol{x})$ 最小化的向量 $\boldsymbol{x} \in \mathbb{R}^n$。用于解决最优化问题的 BCD 算法已经被充分研究[21]。在 BCD 中,$\boldsymbol{x}$ 被分解为 β 个等长块 $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_\beta$;记第 k 轮计算中的向量 $\boldsymbol{x}$ 为 $\boldsymbol{x}^k = [\boldsymbol{x}_1^k, \boldsymbol{x}_2^k, \cdots, \boldsymbol{x}_\beta^k]$。在第 k 轮迭代中,除了被选定的块 $\boldsymbol{x}_i^k$ 外,其余所有块的值都是固定的。更新过程为 $\boldsymbol{x}_i^{k+1} = \boldsymbol{x}_i^k + \alpha_i^k \boldsymbol{d}_i^k$,其中 α_i^k 为步长,$\boldsymbol{d}_i^k$ 为下降方向。

本文中,使用 $G = \{V, E\}$ 表示输入图;V、E 分别为点集和边集;v_i 表示第 i 个点;e_{ij} 表示从点 v_i 到点 v_j 的边;d_i^+、d_i^- 分别表示点 v_i 的出度和入度;$\boldsymbol{A}$ 表示邻接矩阵;N_i^+、N_i^- 分别表示点 v_i 的出邻居集和入邻居集。

下面将图计算任务中的一个经典任务 PageRank 转化为最优化问题,并介绍如何应用 BCD 求解。PageRank 的更新公式为 $\boldsymbol{x}^{k+1} = \boldsymbol{P}\boldsymbol{x}^k + \boldsymbol{b}$,其中 $\boldsymbol{P} = \alpha(\boldsymbol{D}^{-1}\boldsymbol{A})^{\mathrm{T}}$,$\boldsymbol{b} = (1-\alpha)/|V|$,$\boldsymbol{D}$ 为由每个点的出度构成的对角阵,α 为 PageRank 的阻尼系数。当 PageRank 收敛时,记最终结果为 $\boldsymbol{x}^*$,那么有 $\boldsymbol{x}^* = \boldsymbol{P}\boldsymbol{x}^* + \boldsymbol{b}$。由此构建 PageRank 收敛的最优化问题的目标函数,其 $L-2$ 正则形式为 $\min\limits_{\boldsymbol{x}\in\mathbb{R}^n} F(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{P}\boldsymbol{x}^* + \boldsymbol{b} - \boldsymbol{x}^*)^2$ 选择的下降方向 $\boldsymbol{d}$ 为目标函数的梯度。

$$\nabla_{x_i} F(\boldsymbol{x}) = \sum_{j=1}^{|V|} p_{ij} x_j - \frac{1-\alpha}{|V|} \tag{1}$$

其中若 $i = j$ 则 $p_{ij} = 1$;若点 v_i 与 v_j 有连边则 $p_{ij} = -\alpha/d_j^+$;其他情况 $p_{ij} = 0$。令式(1)等于 0,由此得到每轮的更新函数:

$$x_i^{k+1} = \alpha \sum_{j\in N_i^-} \frac{x_j^k}{d_j^+} + \frac{1-\alpha}{|V|} \tag{2}$$

至此,将 PageRank 转化为了使用 BCD 解决的最优化问题。关于图算法的转化以及块选择方法的选择在现有的工作中有详尽的讨论[19-21],包括弱连通分量、最短路径和协同过滤等。

BCD 算法给定 3 个可配置参数,块大小 σ、块选择方法和块更新方法。块大小表示被分配到一个块中的点数,取值范围为 $1 \leqslant \sigma \leqslant |V|$。块选择方法可以是预定义的固定顺序或者是目标函数的梯度方向。块更新方法指定图算法使用的迭代更新函数。若以梯度下降方法为块选择方法,BCD 中的坐标下降方向沿着目标函数的梯度方向。以上述 PageRank

为例,式(1)计算每个点的梯度下降方向,式(2)表示每次迭代的更新函数。坐标下降方向表示给定图的哪些部分对算法的收敛最"有价值",也即能够让目标函数最速下降的方向。

4.2　调度策略

基于4.1节描述的方法,实现了基于BCD的PageRank算法,并在LiveJournal(LJ)和Twitter-2010(TW)数据集上进行了测试。对于任意BCD迭代轮次 k,块选择方法产生的优先块顺序为 $[b_1^k, b_2^k, \cdots, b_\beta^k]$,其中 β 为总块数,块 b_i^k 的优先级大于块 b_{i+1}^k 的优先级,那么在第 k 轮中块 b_1^k 被选中。

定义事件 A:块 b_i^k 在第 $(k+i-1)$ 轮中被选中,即 $b_i^k = b_1^{k+i-1}$。考察事件 A 的频率,结果如图2所示,左右图分别为LJ和TW数据集,每条曲线表示不同的测试块大小,水平和垂直的虚线表示每次块选择方法产生的优先序列中的前 τ 个块与接下来 τ 轮中被选中的块一致的频率超过90%。由此可以得到,每轮的块选择方法可以预测接下来若干轮的选择结果。

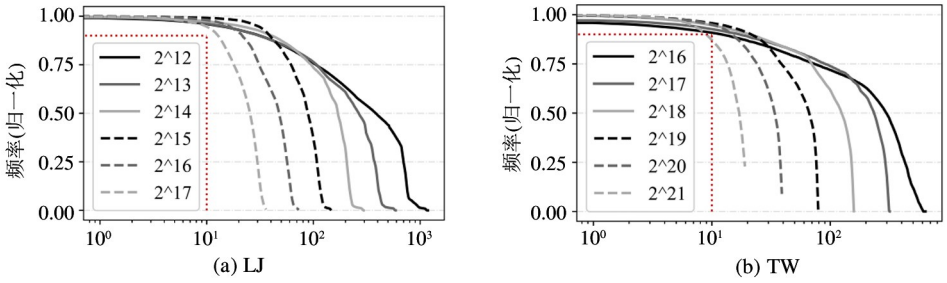


图2　事件 A 发生频率(归一化)

据此,设计BCDG的调度策略为一次选择多轮优先。在BCDG中,为了区分与BSP模型中图计算算法迭代轮次的概念,称从一次选择开始到下一次选择开始前为一个超步(superstep)。记每个超步的块选择个数为 τ。在每个超步开始时,执行块选择方法计算数据块的优先级,选择优先级最大的前 τ 个块加载、计算,如图3中的①所示。那么相较于原始的BCD算法,BCDG节省了 $\tau-1$ 次计算块选择方法的时间,并且能够将加载与计算重叠起来。而考虑到 $[b_1^k, \cdots, b_\tau^k]$ 不能精确地预测 $[b_1^k, \cdots, b_1^{k+\tau-1}]$,$\tau$ 越大会出现越多的冗余计算,并且影响收敛效率。根据图2的结果,推荐选取能够保障预测准确率90%的 $\tau = 5 \sim 10$。

预取策略。图3为调度和预取流水线示意图,$t2$ 时刻前后分别表示超步1和超步2。由于加载队列的任务是根据选择的结果派发,当计算超步中最后一个块以及计算块选择方法时,加载线程必定为空闲,如图3①中的时刻 $t_1 \sim t_3$。为了更好地重叠加载与计算,并且进一步利用SSD的读带宽,引入预取机制,即在计算过程中,当加载任务队列中本轮的 τ 个块加载任务完成时,继续从 $[b_{\tau+1}^k, \cdots, b_{2\tau}^k]$ 中加载。图3②中时刻 $t_1 \sim t_3$ 继续加载块优先列表中的内容,以期在下一个超步中可以命中选择的块。算法1展示在 t_3 时刻(第4行)主线程收到第 $k+1$ 的优先选择结果,如果加载队列中的任务尚未完成,则需要分情况处理:如果正在加载的块不在超步2的选择列表中(第7行),则立即停止加载,并将选择列表中的尚未被加载的块加入加载队列(第9行);否则,如果选择列表中存在尚未被加载、且优先级高于当前块的块(第13行),就停止当前加载,并将这些块加入加载队列(第15行),如果不存在,则继续当前加载。另外,如果加载过程中(包括预取阶段)内存不足,则将已经计算完毕且不在加载队列中的块汰换出内存。

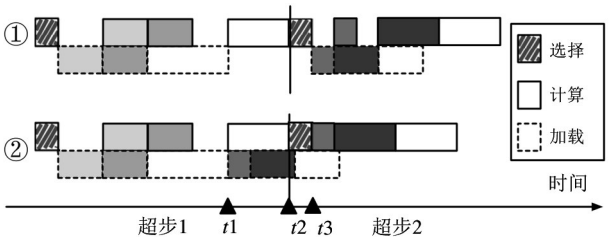


图3　调度和预取流水线(①未使用预取②使用预取)

算法 1　预取策略

功能:根据数据块的优先级处理预取队列

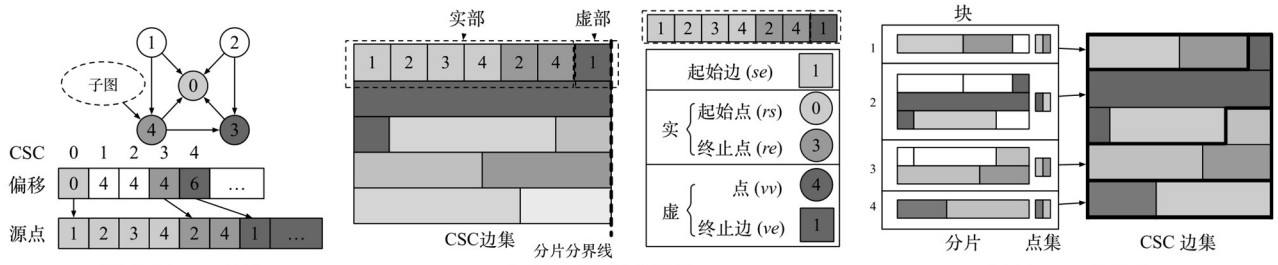
1. //加载线程
2. //Load(b_i^k)　//加载第 k 轮的第 i 个数据块
3. // 主线程
4. **if** Receive(Select($k+1$)) **then**　// t_3 时刻
5. $[b_1^{k+1}, \cdots, b_\beta^{k+1}] \leftarrow$ Select($k+1$)
6. $\mathbb{S} \leftarrow [b_1^{k+1}, \cdots, b_\tau^{k+1}]$　//选择前 τ 个块
7. **if** $b_i^k \notin \mathbb{S}$ **then**　//当前加载的块不在下一轮的优先列表中
8. Send(*Stop*)　//发送停止信号给加载线程
9. EnQueue($\{b \in \mathbb{S} \mid b$ is unloaded$\}$)
10. **else**
11. $\exists 1 \leqslant j \leqslant \tau$, $b_j^{k+1} = b_i^k$ //当前块 i 在下一轮优先列表中位置为 j
12. **for** $1 \leqslant i \leqslant j$ **do**
13. **if** *Block* b_i^{k+1} is unloaded **then**
14. Send (*Stop*)　//发送停止信号给加载线程
15. EnQueue (b_i^{k+1})

5　图数据的划分与计算

本节介绍 BCDG 的计算调度分离的划分策略与其相应的计算模式。根据第 2.2 节的分析,在 BCD 算法中,数据块的大小对于性能的影响很大。更大的块有利于计算局部性和点间的计算并行性,但是牺牲了收敛速度;更小的块整体收敛轮次减少,但是计算性能会降低。在第 4 节中介绍了 BCDG 的调度策略,基于 BCD 算法对图数据根据点集进行了划分,每个数据块包含的点数一致。在图计算任务中,由于图数据的偏斜性,为了负载均衡,往往需要均衡每个数据块的边数量而非点数。因此,BCDG 首先针对图的边数据构建等宽的分片(chunk),然后以分片为最小运算单元进行计算和调度。

5.1　图数据的分片

对于给定图 $G(V, E)$,G 的原始压缩稀疏列结构如图 4(a)所示;图 4(b)展示了在原始的 CSC 数组上的划分策略,左侧每行表示一个分片,不同的灰度表示不同点的邻居,右侧为分片的数据结构。BCDG 中,假设图计算中每条边上需要的计算资源是等价的。为了平衡计算负载,BCDG 将边集列表切分为包含同等数量边的若干部分,每个部分为一个分片。每个分片则为整体图计算的最小运算单元,针对每个分片的计算是通过单线程串行执行的。



(a) 样例图和CSC结构　(b) 分片策略与数据结构　(c) 分片与BCD数据块

图 4　图划分策略与数据块调度

然而,如此切分会在面对相同目标点的时候引入写冲突,如图 4(b)所示,分片分界线将处在分片边界的目标点的邻居划分到了不同分片。为了解决写冲突的问题,引入实虚点机制。在每个分片中,若一点的邻居列表中的最后一个点属于本分片,那么称此点为实点,反之称之为虚点。分别称实点、虚点对应的边集为实部、虚部。那么一个分片 C_i 的逻辑构成如下:

$$C_i \leftarrow \{se_i, Real_i(rs, re), Virt_i(vv, ve)\} \quad (3)$$

其中 *se* 表示起始边,*rs*、*re* 分别表示实部中的起始和终止点,*vv*、*ve* 分别表示虚部中的虚点和其在虚部中的终止边。注意,一个分片中的虚部只包含一个虚点。BCDG 维护分片的集合 $\boldsymbol{C}$。BCDG 维护了每个虚点和其对应的实点的映射:

$$RV \leftarrow \{(r_i, v_i) \mid \forall C_i \in \boldsymbol{C}, Virt_i \neq \text{null}\} \quad (4)$$

那么,在一轮迭代计算中,BCDG 首先将每个虚点添加至计算边缘中,并对虚点进行等价实点的计算、更

新操作;在此轮计算结束前,将每个虚点的更新至合并到对应的实点结果中。

算法 2 描述了基于分片的最小运算单元计算流程 CalChunk。其中第 1 ~ 2 行从输入图中读入 CSC 结构图数据,包含偏移量数组和数据数组(图 4(a))。它串行计算了单个分片的实部和虚部,并分别更新了对应的计算边缘(frontier)值。函数 Comp 的输入值为每条边的源点和目标点,用于执行在每条边上的计算。函数 Filter 作为计算中的可选项,其作用为自定义过滤计算时的不必要计算的点或边。并行计算时,首先将虚点扩展至计算边缘,然后每轮迭代中利用 CalChunk 并行计算每个分块的内容,利用用户定义的 Reduce 函数在上 RV 计算,将虚点的结果合并至其对应的实点结果中,并更新计算边缘,重复此过程直至收敛。

算法 2　最小运算单元 CalChunk

输入:分片 C_i
输入:边计算函数 Comp(src,dst)
输入:过滤器 Filter(src,dst)
输出:单个分片串行运算函数 CalChunk

1. $\boldsymbol{s} \leftarrow \{s_i \mid \forall (s_i, d_i) \in E, d_i \leqslant d_{i+1}\}$　//CSC 数据数组
2. $\boldsymbol{o} \leftarrow \{o_i = \mid N_i^- \mid + o_{i-1} \mid \forall i \in V\}$　//CSC 偏移量数组
3. **Func** CalChunk(C_i, Comp, Filter):
4. **if** $Real_i \neq$ null **then**
5. **for** $rs \leqslant v \leqslant re$ **do**　// 计算实部所有点
6. **for** $s_e \leqslant u < s_{o_v}$ **do**　// 遍历点 v 的所有邻居
7. **if** Filter(u, v) **then**
8. $x'_v \leftarrow$ Comp(u, v) //更新 v 对应的计算边缘
9. **if** $Virt_i \neq$ null **then**
10. $r, v \leftarrow RV_i$　// 获取虚部的实虚点对
11. **for** $s_e \leqslant u < s_{ve}$ **do**　// 遍历虚点对应的邻居
12. **if** Filter(u, r) **then**
13. $x'_v \leftarrow$Comp(u, v) //更新 v 对应的计算边缘

5.2　基于分片的块计算

为了应用 BCD 算法于图计算,BCDG 以分片作为粒度将给定的图划分为若干个块,然后按照分片运行基于 BCD 的图算法。图 4(c)展示了图划分的设计。对于 BCD 视角,点集被分割为 β 个片段,其中 β 由 BCD 算法的块大小确定。据此,给定图被划分为 β 个子图。一个点集片段 V_{B_i} 和其对应的边集 E_{B_i} 构成子图 B_i,其中 $1 \leqslant i \leqslant \beta$。以分片作为计算粒度,因此子图 B_i 由分片集合 $\boldsymbol{C}_{B_i}$ 构成,其包含所有 E_{B_i} 中的边。$\boldsymbol{C}_{B_i}$ 满足下述条件:$V_{B_i} \subseteq \cup_{c \in \boldsymbol{C}_{B_i}} V_c$,$E_{B_i} \subseteq \cup_{c \in \boldsymbol{C}_{B_i}} E_c$,其中 V_c 表示分片 c 中的点集,包括真实点和其对应的虚拟点;E_c 表示分片 c 中的边集。在 BCD 计算中,BCDG 依据块选择方法在初始迭代轮次选择 τ 个块,其中 $1 \leqslant \tau \leqslant s$。图计算的计算边缘由这些被选中的块中的点集构成。然后,BCDG 以分片作为粒度依次遍历被选中的子图中的所有边。

算法 3 描述了基于分片的块计算流程。其中第 1 ~ 2 行为 BCD 算法中的特殊过滤器 BlockFilter 函数,其消除了不属于被选中块的真实点以及其对应的虚拟点。第 3 行初始化计算边缘,并将虚拟点扩展至计算边缘。第 4 ~ 12 行为迭代计算过程,重复计算直到收敛。第 5 行使用 SelectBlock 函数选择一个或多个块。第 7 ~ 8 行并行计算分片。第 9 ~ 10 行将所有虚拟点的结果合并至其对应的真实点结果中。当所有优先块计算完毕后,第 11 行扩展计算边缘。

算法 3　基于分片的块并发计算

输入:分片后 $\{G(V, E), \boldsymbol{C}, RV\}$
输入:边计算函数 Comp(src,dst)
输入:实虚点合并函数 Reduce(real,virt)
输入:过滤器 Filter(src,dst)
输入:块数 β, 选择块数 τ
输出:收敛的计算结果 $\boldsymbol{xv}^*$

1. **Func** BlockFilter(B_i, s, d):　// 过滤非块 B_i 的点
2. **return** ($d \in V_{B_i}$) ? Filter(s,d): false
3. Frontier $\boldsymbol{xv} \leftarrow [x_1, \cdots, x_{|V|+|RV|}]^{\mathrm{T}}$　// 初始化计算边缘
4. **repeat**
5. $[B_1, \cdots, B_\tau] \leftarrow$ SelectBlock()　//计算得优先选择列表
6. **for** $1 \leqslant i \leqslant \tau$ **do**　// 遍历计算每个块
7. **Parallel for** $\forall C \in \boldsymbol{C}_{B_i}$ **do**
8. CalChunk(C, Comp, BlockFilter(B_i))
9. **Parallel for** $r, v \in RV_{B_i}$ **do**
10. $x'_r \leftarrow$ Reduce(r, v)
11. $\boldsymbol{xv} \leftarrow [x'_1, \cdots, x'_{|V|}, 0, \cdots, 0]^{\mathrm{T}}$　// 将虚点对应值置空
12. **until** Converged

6 实 验

6.1 实验设置

实验环境:单台计算节点,CPU 为 Intel(R) Xeon(R) E5-2640 v4 CPU(双节点; 40 个超线程),128 GB内存,1 块 477 GB 的 SSD(读写带宽分别为 3.0 GB/s 和 2.7 GB/s)。

基线:GridGraph[8]是外存图处理系统的一个强有力的基线。Lumos[3]是基于 GridGraph 实现的框架,其提供了对十亿级图处理的同步执行框架。由于实验环境中的计算单元是 CPU,所以本文复现了 GraphABCD 的 CPU 版本作为基线。在第 6.2 节中对比了 GraphABCD 文中的基线 GraphMat[22],结果见表 1,可以看出复现的结果与文中报告的性能提升一致,因此可以以此来作为基线。

表 1 BCDG 与基线的运行时间(s)对比

Dataset		PO	LJ	TW	RM	YH
PR	BC	0.19	0.26	7.34	6.83	92.86
	GG	2.99	3.77	68.74	45.96	749.66
	LU	2.09	3.26	53.43	39.48	/
	GA*	0.65	1.10	28.66	26.82	-
	GM	1.46	3.84	54.38	-	-
CC	BC	0.07	0.17	10.24	9.28	157.24
	GG	0.34	0.88	21.11	30.49	20 840
	GA*	0.33	0.61	19.18	13.72	-

数据集:使用 4 个真实世界的图数据和 1 个合成图数据,如表 2 所示,其中 Pokec[23](PO)、LiveJournal[24](LJ)和 Twitter-2010[25](TW)为社交网络图,Yahoo[26](YH)为网页链接图,RMat27[27](RM)为使用幂律分布合成的图数据。文件大小为二进制边集文件大小,图计算系统一般存储CSC、CSR结构各一份,与二进制边集文件大小相近。

表 2 实验中使用的数据集

数据集	点数	边数	文件大小
Pokec	1.63×10^6	30.62×10^6	244 MB
LiveJournal	4.85×10^6	68.99×10^6	527 MB
Twitter-2010	41.65×10^6	1.47×10^9	12 GB
Yahoo	1.41×10^9	8.05×10^9	64 GB
RMat27	134.00×10^6	2.12×10^9	17 GB

6.2 与现有工作比较

将 BCDG 与 GridGraph(GG)、Lumos(LU)和复现的 GraphABCD(GA*)以及其基线 GraphMat(GM)进行对比,在表 2 中的数据集上测试了 PageRank 应用直到收敛于相同的条件。每次测试均使用全部 40 个超线程,运行 10 次取平均结果。表 1 报告了测试的结果,"-"表示超内存,"/"表示无法运行。其中复现的 GraphABCD 和 GraphMat 在纯内存中运行,BCDG 与 GridGraph、Lumos 运行中均使用了 SSD。对于外存测试,使用了 cgroup 限制程序运行内存。在所有数据集上,BCDG 的性能均优于其他系统。

对比 GraphABCD。首先,复现的 GraphABCD 相较于 GraphMat 性能平均提升 2.46 倍,与其报告的 2.1~2.5 倍一致。除了在 Yahoo 数据集上,由于内存不足复现的 GraphABCD 无法完成,在其余数据集上,相较于复现的 GraphABCD,BCDG 性能平均提升了 3.86 倍(PageRank)和 2.61 倍(CC)。由于 BCDG 的调度策略为选择更多的数据块而非一个,所以相比于 GraphABCD,每轮计算重新选择块提升了计算性能。

对比 GridGraph 和 Lumos。在不限制内存的情况下,BCDG 相比 GridGraph 性能平均提升了 10.30 倍(PageRank)和 3.61 倍(CC),相比 Lumos 提升了平均 8.72 倍(PageRank)。由于 GridGraph 在每轮迭代中需要加载全量数据,并且每个数据块在每轮计算中只计算一次,外存 I/O 时间为主要开销。虽然 Lumos 中使用了异步的计算逻辑,但是其载入的每个数据块只计算 2 次,在一定程度上缓解了加载开销,但这仍然是瓶颈。BCDG 相比于 GridGraph 的提升主要体现在两方面,一方面优先加载策略节省了计算中的外存加载总量,从而节省了外存开销;另一方面基于分片的划分方式更加均衡了计算负载。

表 3 报告了限制内存时,BCDG 与 GridGraph、Lumos 在数据集 LJ 和 TW 上运行 PageRank 的时间对比,第 1 列和第 4 列表示在 LJ 和 TW 上限制内存的大小,单位为字节,"-"表示不限制内存,"/"表示无法运行。当限制内存时,GridGraph 和 Lumos 的性

表 3　限制内存时运行时间(s)对比

LJ	BC	GG	LU	TW	BC	GG	LU
-	0.26	3.77	3.26	-	7.34	68.74	53.43
2 GB	0.30	3.69	10.14	8 GB	8.96	68.28	53.55
1 GB	0.31	3.72	10.15	6 GB	11.75	68.43	53.30
768 MB	0.37	/	/	4 GB	16.41	68.35	54.88
512 MB	0.59	/	/	2 GB	21.13	/	/

能没有明显变化。BCDG 性能损失较大,因为内存较小时,BCDG 会选择更小的 BCD 块放入内存中计算;另外,内存较小时预取策略几乎失效。但当限制更小的内存上限时,只有 BCDG 能够正常运行,这是因为 BCDG 的分片策略使得调度计算能够以分片作为最小粒度,其相对于 GridGraph 的二维划分而言粒度更细。当内存充足时,BCDG 的预取策略可以有更大的空间存储对下一个超步计算的数据块的预测。

6.3　调度策略

图 5 描述了 BCD 算法中选择不同的 BCD 块数时,块选择个数 τ 对于收敛时间的影响,左右图分别为数据集 LJ 和 TW 上的结果,图中横轴为选择块的个数 τ, 每条曲线表示总 BCD 块数。注意,块大小 σ 与 BCD 块数的积约等于图的点数。从图中可得,在 2 个数据集上,每种块大小的选择下,相比于原始 BCD 算法中只选择一个块,增加选择块数都可使收敛时间显著减少。其原因为选择多个块使得加载和计算能够重叠起来,而其预测块的性能损失远小于重叠带来的性能提升。当选择块数 $\tau > 5$ 时,收敛时间的降低速度有所放缓。其原因为当选择块数增加到一定程度时,流水线重叠趋于稳定,选择更多的块不会更显著地降低 CPU 等待时间。当选择块数 $\tau > 10$ 时,收敛时间普遍出现波动。其原因为选择更多的块意味着预测正确率下降,所以出现了较多的冗余计算。根据实验结果,推荐选取能够保障预测准确率 90% 的块选择数 τ 为 5 ~ 10。

6.4　预取策略

图6描述了当选择块数为5时,预取的块数对

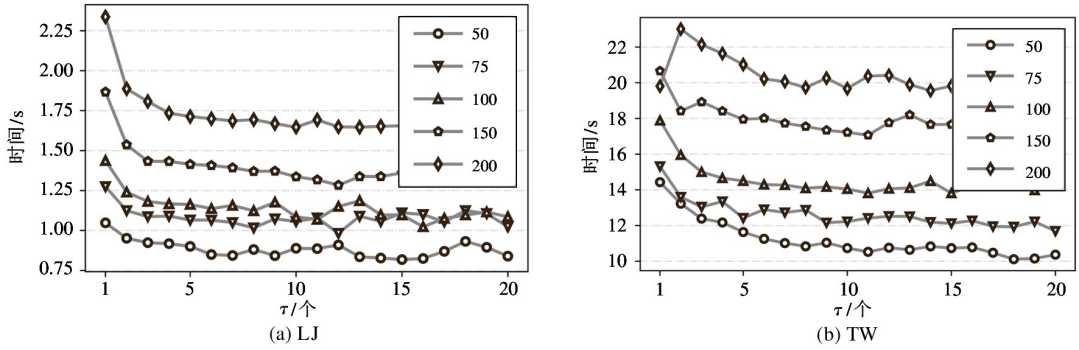


图 5　不同 BCD 数据块大小下 PageRank 收敛时间统计

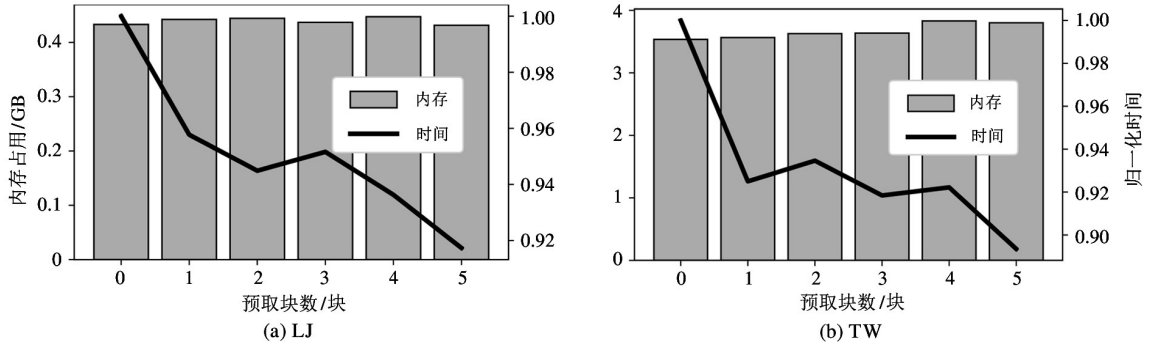


图 6　预取块数对运行时间和消耗内存的影响

运行时间与消耗内存的影响,左右图分别为数据集 LJ 和 TW 上的结果,横轴表示预取的块数,左纵轴表示内存消耗,右纵轴表示归一化的运行时间。其中测试应用为 PageRank,运行时间按照关闭预取策略的对照组(预取块数为 0)进行归一化。从图中得出结论:预取更多的块几乎不会造成更多的内存消耗,但是却可以显著降低整体的运行时间,当预取块数与选择块数一致时,运行时间总体降低 10%。这得益于预取策略能够更好地利用加载线程,消除其在图 3 中 $t_1 \sim t_3$ 时间内的空闲。

6.5 运行时间分析

图 7 描述了 BCDG 的运行时间分析。其中测试应用为 PageRank,图 7 给出了不同数据集在不限制内存和限制内存情况下的运行时间分析,在数据集 PO、LJ、TW 和 RM 上的内存限制分别为 256 MB、512 MB、4 GB 和 6 GB,运行时间按照每个例子的总运行时间进行归一化。从图中可以得出,相比于 GridGraph 中外存 I/O 占总计算时间的 70% ~ 80%、占 Lumos 的 50% ~60%[3],BCDG 从外存读入的时间在总计算时间的占比仅为 10% ~30%。限制内存时,BCDG 的性能一方面受到 BCD 选块大小的影响,另一方面也需要消耗更多的时间在加载数据上。这是因为预取策略受到内存限制,优化性能有限。另外也说明了相比于其他外存图计算系统,BCDG 能够更加高效地利用内存减少外存 I/O。

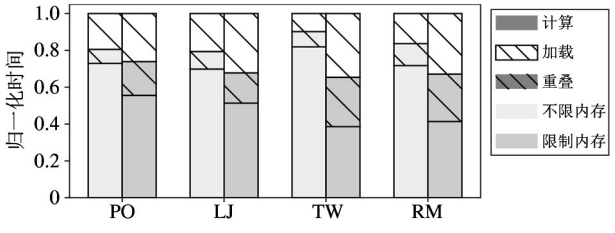


图 7 不同数据集上运行 PageRank 的运行时间分析

7 结 论

本文针对外存图计算系统存在的计算冗余问题以及外存 I/O 瓶颈,在基于优先选择的外存异步图计算系统上作了深入研究。计算数据块优先级本身引入的性能开销不可忽略,并且优先选择策略会破坏加载计算的流水线以及会引入计算的负载不均衡。为此本文提出了基于块坐标下降法的外存异步图计算系统。首先,观察并验证可得,BCD 算法在每轮中赋予每个数据块的优先级能够预测后续轮次的选择,由此设计了一次选择多轮优先调度策略;其次,继续利用此性质设计了预取策略,并构建加载计算流水线;最后,发现面向计算的按边划分策略不影响按点调度,因此设计了计算调度分离的划分策略,同时实现了计算的负载均衡和优先调度。实验结果表明,相比于目前最先进的外存图计算系统 GridGraph 和 Lumos,BCOG 平均性能分别提升 10.30 倍与8.72倍。整体计算过程中,CPU 等待外存 I/O 的时间仅占 10% ~30%。本文更多关注基于推拉计算模型中的拉操作,后续工作将对推操作进一步探索。

参考文献

[1] ZHU X, CHEN W, ZHENG W, et al. Gemini: a computation-centric distributed graph processing system[C]// Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, USA, 2016: 301-316

[2] LIU H, HUANG H H. Graphene: fine-grained IO management for graph computing[C]// The 15th USENIX Conference on File and Storage Technologies, Santa Clara, USA, 2017:285-300

[3] VORA K. LUMOS: dependency-driven disk-based graph processing[C]//2019 USENIX Annual Technical Conference, Renton, USA, 2019:429-442

[4] SHUN J, BLELLOCH G E. Ligra: a lightweight graph processing framework for shared memory[C]// Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Shenzhen, China, 2013: 135-146

[5] AI Z, ZHANG M, WU Y, et al. Squeezing out all the value of loaded data: an out-of-core graph processing system with reduced disk I/O[C]//2017 USENIX Annual Technical Conference, Santa Clara, USA, 2017:125-137

[6] ZHUO Y, CHEN J, LUO Q, et al. SympleGraph: distributed graph processing with precise loop-carried dependency guarantee[C]// Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, London, UK, 2020:592-607

[7] KYROLA A, BLELLOCH G, GUESTRIN C. GraphChi: large-scale graph computation on just a PC[C]// The 10th USENIX Symposium on Operating Systems Design and Implementation, Hollywood, USA, 2012:31-46

[8] ZHU X, HAN W, CHEN W. Gridgraph: large-scale graph processing on a single machine using 2-level hierarchical partitioning[C]//2015 USENIX Annual Technical Conference, Santa Clara, USA, 2015:375-386

[9] 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

[10] FALOUTSOS M, FALOUTSOS P, FALOUTSOS C. On power-law relationships of the internet topology[J]. *ACM*

SIGCOMM Computer Communication Review, 1999, 29(4): 251-262
[11] VORA K, XU G, GUPTA R. Load the edges you need: a generic I/O optimization for disk-based graph processing[C]//2016 USENIX Annual Technical Conference, Denver, USA, 2016: 507-522
[12] KUMAR P, HUANG H H. Graphone: a data store for real-time analytics on evolving graphs[J]. *ACM Transactions on Storage* (*TOS*), 2020, 15(4): 1-40
[13] ZHANG M, WU Y, ZHUO Y, et al. Wonderland: a novel abstraction-based out-of-core graph processing system[J]. *ACM SIGPLAN Notices*, 2018, 53(2): 608-621
[14] CHENG S, ZHANG G, SHU J, et al. Asyncstripe: I/O efficient asynchronous graph computing on a single server[C]//Proceedings of the 11th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Pittsburgh, USA, 2016: 1-10
[15] 周晓丽, 陈榕. 面向图计算系统的异步计算-加载模型[J]. 小型微型计算机系统, 2019, 40(6): 1221-1226
[16] 张程博, 李影, 贾统. 面向分布式图计算作业的容错技术研究综述[J]. 软件学报, 2021, 32(7): 2078-2102
[17] De LEO D, BONCZ P. Teseo and the analysis of structural dynamic graphs[J]. *Proceedings of the VLDB Endowment*, 2021, 14(6): 1053-1066
[18] ZHAO C, ZHANG Z, XU P, et al. Kaleido: an efficient out-of-core graph mining system on a single machine[C]//2020 IEEE 36th International Conference on Data Engineering IEEE, Dallas, USA, 2020:673-684
[19] ZHANG Y, GAO Q, GAO L, et al. PrIter: a distributed framework for prioritized iterative computations[C]//Proceedings of the 2nd ACM Symposium on Cloud Computing, Cascais, Portugal, 2011: 1-14
[20] ZHANG Y, GAO Q, GAO L, et al. Maiter: an asynchronous graph processing framework for delta-based accumulative iterative computation[J]. *IEEE Transactions on Parallel*, *Distributed Systems*, 2013, 25(8): 2091-2100
[21] YANG Y, LI Z, DENG Y, et al. GraphABCD: scaling out graph analytics with asynchronous block coordinate descent[C]//2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture, Valencia, Spain, 2020: 419-432
[22] SUNDARAM N, SATISH N, PATWARY M M A, et al. GraphMat: high performance graph analytics made productive[J]. *Proceedings of the VLDB Endowment*, 2015, 8(11): 1214-1225
[23] TAKAC L, ZABOVSKY M. Data analysis in public social networks[C]//International Scientific Conference and International Workshop Present Day Trends of Innovations, Lomza, Poland, 2012: 1-6
[24] 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
[25] KWAK H, LEE C, PARK H, et al. What is Twitter, a social network or a news media? [C]//Proceedings of the 19th International Conference on World Wide Web, Raleigh, USA, 2010:591-600
[26] Yahoo! [EB/OL] https://webscope.sandbox.yahoo.com/: Yahoo,[2021-09-17]
[27] CHAKRABARTI D, ZHAN Y, FALOUTSOS C. R-MAT: a recursive model for graph mining[C]//Proceedings of the 2004 SIAM International Conference on Data Mining, Lake Buena Vista, USA, 2004:442-446

Block coordinate descent based asynchronous out-of-core graph computing system

ZHAO Cheng, ZHANG Zhibin, GUO Jiafeng, LIU Dingwei
(Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
(University of Chinese Academy of Sciences, Beijing 100049)

Abstract

For existing out-of-core graph computing systems, insufficient external memory I/O bandwidth is a performance bottleneck. Using a bulk synchronous parallel model will lead to redundant computation, while using an asynchronous parallel mode introduces additional priority computation overhead and load imbalance. In this paper, a block coordinate descent (BCD) based asynchronous out-of-core graph computing system (BCDG) is proposed. A scheduling strategy named one-selection multi-round priority is designed to reduce the average overhead of priority computation; a priority-based block prefetching strategy is designed to solve the problem that priority selection destroys the sequential execution pipeline; a partitioning strategy to separate computation and scheduling is designed to achieve balanced edge-by-edge computation and ensure vertex-by-vertex scheduling. The experimental results show that compared with two state-of-the-art out-of-core graph computing systems, GridGraph and Lumos, the proposed system outperforms them by 10.30 times and 8.72 times on average respectively. During the overall computation, the central processing unit (CPU) spends only 10% – 30% of its time waiting for disk I/O.

Key words: graph computing system, out-of-core, asynchronous parallel computing, block coordinate descent (BCD), graph partition