计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20131343
52(4):960-971,2015

BSP 模型下基于边聚簇的大图划分与迭代处理

冷芳玲[1] 刘金鹏[1] 王志刚[1] 陈昌宁[1] 鲍玉斌[1] 于 戈[1] 邓 超[2]
[1](东北大学信息科学与工程学院 沈阳 110819)
[2](中国移动通信研究院业务支撑研究所 北京 100053)
(liujinpengatneu@163.com)

Edge Cluster Based Large Graph Partitioning and Iterative Processing in BSP

Leng Fangling[1], Liu Jinpeng[1], Wang Zhigang[1], Chen Changning[1], Bao Yubin[1], Yu Ge[1], and Deng Chao[2]
[1](*College of Information Science and Engineering*, *Northeastern University*, *Shenyang* 110819)
[2](*Division for Business Support*, *China Mobile Institute*, *Beijing* 100053)

Abstract With the development of Internet and the gradual maturity of related techniques in recent years, the processing of large graphs has become a new hot research topic. Since it is not appropriate for traditional cloud computing platforms to process graph data iteratively, such as Hadoop, researchers have proposed some solutions based on the BSP model, such as Pregel, Hama and Giraph. However, since graph algorithms need to frequently exchange intermediate results in accordance with the graph's topological structure, the tremendous communication overhead impacts the processing performance of systems based on the BSP model greatly. In this paper, we first analyze the solutions proposed by the well-known BSP-based systems in reducing communication overhead, and then propose a graph partition strategy named edge cluster based vertically hybrid partitioning (EC-VHP), building a cost benefit model to study its effectiveness to the communication overhead. Then based on EC-VHP, we propose a vertex-edge computation model, and design both a plain hash index structure and a multi-queue parallel sequential index structure to further improve the processing efficiency of message communication. Finally, our experiments on real and synthetic data sets demonstrate the efficiency and accuracy of the EC-VHP and the index mechanism.

Key words large graph; BSP model; graph partition; vertex-edge computation model; index structure

摘 要 近年来随着互联网的普及和相关技术的日益成熟,大规模图数据处理成为新的研究热点.由于传统的如 Hadoop 等通用云平台不适合迭代式地处理图数据,研究人员基于 BSP 模型提出了新的处理方案,如 Pregel,Hama,Giraph 等.然而,图处理算法需要按照图的拓扑结构频繁交换中间计算结果而导致巨大的通信开销,这严重地影响了基于 BSP 模型的系统的处理性能.首先从降低消息通信的角度分析当前主流 BSP 系统的处理方案,然后提出了一种基于边聚簇的垂直混合划分策略(EC-VHP),并建立代价收益模型分析其消息通信优化的效果.在 EC-VHP 的基础上,提出了一个点-边计算模型,并设计了简单 Hash 索引和多队列并行顺序索引机制,进一步提高消息通信的处理效率.最后,在真实数据集和模拟数据集上的大量实验,验证了 EC-VHP 策略和索引机制的正确性和有效性.

关键词 大规模图;BSP 模型;图划分;点-边计算模型;索引结构

中图法分类号 TP311.13

收稿日期:2013-10-12;**修回日期**:2014-03-05

基金项目:国家自然科学基金重点项目(61033007);国家自然科学基金项目(61173028,61272179);中央高校基本科研业务费专项基金项目(N100704001);教育部-中国移动科研基金项目(MCM20125021)

随着互联网技术的普及和新兴社交网络等应用的快速发展,大规模图数据的高效处理已经成为学术界和工业界的研究热点.真实图数据集尤其是基于互联网的图数据,其规模通常可以达到数十亿顶点和上百亿条边,如万维网的 Web 页面关联图有 5 000 亿顶点和 1 万亿条边,全球最大的社交网站 Facebook 有近 80 亿个顶点和 1 040 亿条边.显然,传统的单机串行和小规模集群都无法处理如此庞大的数据.因此,如何高效地处理大规模图数据成为亟待解决的问题.

Google 公司提出的 MapReduce[1] 计算模型提供了一种简单有效的工作流式处理模式,适用于通用的大数据计算问题,并且目前的 MapReduce 开源实现 Hadoop[2] 已经能够提供商用计算,支持对大规模数据的批处理计算[3].但是对于图算法、图挖掘以及机器学习等需要进行多次迭代的计算,MapReduce 模型并不适用[4].

许多研究者已经证实,在图处理算法中数据并行化的缺失是导致 MapReduce 模型不适合迭代计算的根本原因[4-6].近年来,BSP 模型[5] 在大图处理领域备受推崇,基于 BSP 模型的图处理系统也有很多,主要有 Google 的 Pregel[5]、开源组织 Apache 的 Hama[7] 和 Giraph[8].而随着基于 BSP 模型的各种图处理系统的日臻成熟,这些系统的处理性能以及可靠性等核心问题引起了研究者的关注,其中图计算迭代过程中中间计算结果的频繁网络交换对系统性能的制约尤为显著.

现有的基于 BSP 模型的图处理系统的迭代计算通常是消息驱动、以顶点为中心进行的,在单次迭代中会产生大量的消息而造成网络通信的负担.例如,对于 PageRank 计算,当采用 BSP 系统进行处理时,一次迭代的消息量等价于整个图的边的规模,而对图数据进行有效的划分是降低跨分区的消息规模的有效手段[9].既保证图数据的等规模划分又使得跨分区的边的数目最少(即减少边切割)的数据划分方法在理论上虽然是 NP 难问题,但已经有研究者提出了这一问题的近似解决方案,并且已经存在划分性能良好的软件包供开发者使用[10-11].而对于大多数实际应用而言,保证最少边切割的数据均衡划分通常是不存在的[12].特别是在社交网络中,社区结构覆盖和高密度连通域的存在使得合理划分更具有挑战性[13-15].因此,虽然基于 Hash 的划分方法由于严重地破坏了原始数据的拓扑结构而在并行计算过程中造成巨大的通信开销,但很多研究者仍因其简单高效而提倡采用 Hash 的数据划分方式[16].

简单的水平数据划分(即以图顶点邻接表为单位的横向划分)难以有效解决基于 BSP 模型的系统的消息通信问题,而在 Facebook,Twitter 等社交网络图中,某个顶点的出度边规模可能达到千万量级,这会造成以顶点为中心的计算效率的降低甚至难以顺利进行.因此,研究者着手从边划分和顶点备份的角度给出减少消息通信和减轻点计算负载的解决方案.Giraph[8] 通过在迭代过程中动态优化数据划分来降低通信量,但动态的调整代价过高.GPS[17] 对出度邻接表进行重新调整,但调整策略针对图顶点的整个出度邻接表,不能有效控制调整粒度.而基于 GAS 模型的 PowerGraph[18] 提出了一种基于边的数据备份策略来降低通信开销,但其在图数据备份的处理过程中需要维护一个庞大而复杂的全局信息表,而全局信息表的计算、维护代价过高.

针对上述问题,本文根据数据划分过程中顶点邻接表中的多条边可按照目的顶点所在数据分区进行聚集的特性,提出了基于边聚簇的垂直混合划分策略,在数据加载过程中通过对顶点的邻接表进行 Hash 水平划分与边聚簇垂直混合划分的结合,将满足条件的边划分至同一任务分区,那么在迭代过程中将只向该分区发送 1 条聚簇消息,以压缩消息通信的规模,同时降低点计算的处理负担,提高系统的处理性能.采用 Hash 水平划分保证数据划分的简单高效,边聚簇垂直划分保证降低消息通信规模,且本文的垂直混合划分策略与具体的水平划分方式无关.

本文的主要贡献如下:

1) 提出了基于边聚簇的垂直混合划分(EC-VHP)策略,通过对每个顶点的邻接表进行压缩处理,可以有效地降低消息通信的规模;

2) 基于 EC-VHP 提出点-边计算模型,并设计了简单的 Hash 索引与多队列并行顺序索引的机制,进一步优化消息处理性能;

3) 建立代价估计模型,系统地分析在不同划分粒度下 EC-VHP 策略对系统处理性能的影响.

1 BSP-VHP 系统简介

由东北大学与中国移动联合开发的 BC-BSP 系统是基于 BSP 模型的开源大图迭代处理系统,支持多种数据输入方式和磁盘操作,具有良好的容错控制能力和可伸缩性.本文以 BC-BSP 系统为基础,设计

并实现了 BSP-VHP 系统,后者支持垂直混合划分策略、点边计算模型以及两种索引机制.为了更好地理解本文的相关工作,本节将概括介绍 BSP-VHP 系统的主要功能组件.

1.1 数据模型

BC-BSP 系统以邻接表格式组织输入的原始图数据.本文中,设 $G=(E,V)$ 表示输入的有向图,其中 $V=\{v_1,v_2,\cdots,v_n\}$ 表示图中所有顶点的集合,$E=\{e_1,e_2,\cdots,e_m\}$ 表示图中所有边的集合.图 1(a)描述了一个有向图结构示例,在 BSP-VHP 系统中,图数据仍然采用邻接表的存储格式,如图 1(b)所示:

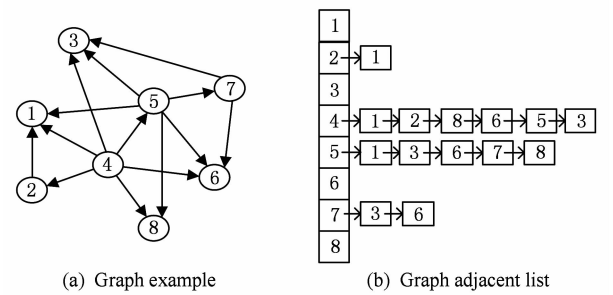


Fig. 1 A graph example and its storage structure.

图 1 图示例及其存储格式

1.2 系统结构

图 2 给出了 BSP-VHP 系统的体系结构,其组件主要包括数据划分控制器、路由控制器、核心计算引擎、消息管理器、任务负载均衡调度器以及高可靠性(HA)控制器.本文工作主要围绕图数据的划分和点计算负载迁移进行展开,对应图 2 中深色模块.其他模块详见文献[19]介绍.

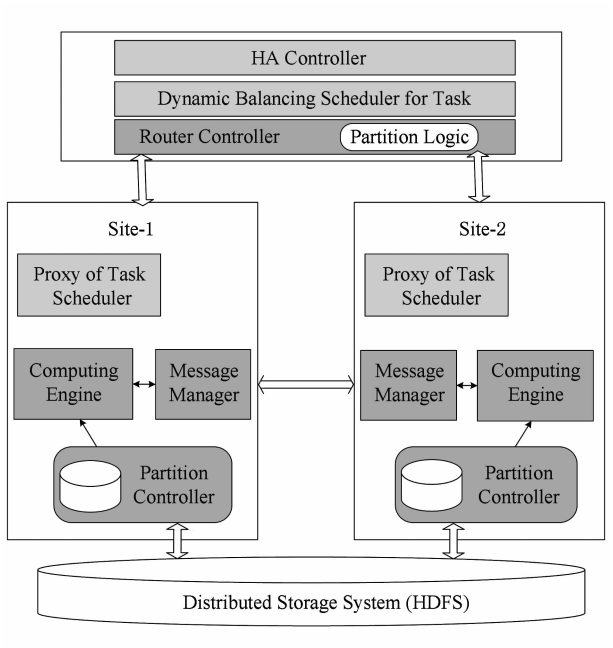


Fig. 2 The system architecture of BSP-VHP.

图 2 BSP-VHP 系统结构

1.2.1 数据划分控制器

用户提交 BSP 作业后,负责处理该作业的部分任务作为实际数据加载器首先从分布式文件系统(如 HDFS)上加载数据到本地,然后根据指定的数据划分策略将数据划分到对应任务的数据分区.

BSP-VHP 系统采用的基本数据水平划分方式是 Hash 划分.水平 Hash 划分方式执行效率高,实现简单,但是会严重破坏原始图的拓扑结构,造成大量的边切割,加剧了跨任务数据分区的通信负担.而通过对图顶点邻接表的垂直划分,将满足划分条件的多条边压缩成一个聚簇边,同时将部分计算负载迁移到聚簇边所在的任务分区,其可以解决这一问题.

虽然 BSP-VHP 系统的实现采用的数据水平划分方式是 Hash 划分,但本文提出的基于边聚簇的数据垂直混合划分策略是与图数据的水平划分方式无关的.这种垂直混合的数据划分能够有效降低网络中的消息通信规模,提升 BSP-VHP 系统的处理性能.

1.2.2 路由控制器

基于 Hash 的水平数据划分保证了 BSP-VHP 系统消息路由计算的高效性能,其不必如 Mizan[20] 一样为维护一个大的路由信息结构而花费巨大代价.然而,由于 BSP-VHP 系统对数据进行了垂直混合划分,不能再使用原有的简单路由计算来进行消息收发.因此,文中 2.1 节引入了聚簇边路由的概念.

从通信的角度,聚簇边路由实质是对满足划分约束的部分边进行路由压缩,多条边消息的路由计算压缩成单个聚簇边路由信息,从而降低了通信的代价.

1.2.3 消息管理器

消息管理器需要维护 3 个消息队列,包括本地任务分区正在处理的消息队列、接收其他任务分区发送消息的消息接收队列以及向其他任务分区发送的消息队列.为了提高处理器的使用率,消息的收发与计算处理采用多线程的处理机制.因为系统对输入的图数据执行了垂直切分,所以对产生的聚簇边路由消息需要进行特殊的处理.其主要处理方案有 2 种:串行的集中式处理和并行的即时处理.两种处理方案对系统的内存利用和通信开销都有不同影响,本文第 3 节对此作进一步的分析讨论.

1.2.4 核心计算引擎

因为 BC-BSP 系统采用以点为中心的计算模型,所以本地计算的计算引擎在每次迭代时需要对本任务数据分区按图顶点进行 1 次遍历计算. 而 BSP-VHP 系统在垂直混合划分基础上使用点-边计算模型,计算引擎划分为点计算与边计算,具体的计算过程与实现的方式有关,详细内容见本文第 2 节提出的点-边计算模型,以及第 3 节针对提高计算的并行处理程度而进行的分析.

2 基于边聚簇的垂直混合划分

基于 BSP 模型的系统在处理大规模图数据时会产生大规模的消息通信,而消息规模对处理器计算、网络通信以及内存的管理均有影响,因此控制消息规模是提升系统性能的重要方面. 本节首先提出了基于边聚簇的垂直混合划分策略,并在其基础上设计了点边计算模型,最后通过建立代价收益模型分析了不同划分粒度对系统性能的影响.

2.1 划分策略

首先定义一些相关概念.

定义 1. 聚簇边. 设任一顶点 v,其出边集合 $N_v=\{e_1,e_2,\cdots,e_k\}$,则 $\Phi(N_v,\psi(\chi))=\{E_1,E_2,\cdots,E_m\}$为顶点 v 的一组聚簇边,当且仅当:

1) 对 $\forall E_j\in\Phi$,$E_j\subseteq N_v$,且 $E_j\neq\varnothing$;

2) $\bigcup\limits_{E_i\in\Phi}E_i=N_v$;

3) 对 $\forall E_i,E_j\in\Phi$ 且 $i\neq j$,有 $E_i\cap E_j=\varnothing$.

产生聚簇边的过程叫作边聚簇. 定义中 ψ 为顶点 v 的边聚簇映射,$\psi:N_v\rightarrow\Phi$. χ 为边聚簇映射约束. 为保证垂直划分有效性,ψ 一般与水平划分的映射规则保持一致.

定义 2. 基于边聚簇的垂直混合划分. 给定一个有向图 $G=(E,V)$、目的任务分区 j 的聚簇边列表集合 G''_j 和一个划分阈值 θ,取 $\forall v\in V$,对于 $\forall E\in\Phi(N_v,\psi)$,若 $|E_i|\geqslant\theta$,则 $\exists t_j\in T$,使得 $assign(E_i)\rightarrow t_j$,同时创建聚簇边列表(clustered edges list, CEL),满足 $CEL(t_j,v)=(v,E_i)$,并且 $G''_j\leftarrow G''_j\cup\{CEL\}$,称这个过程为基于边聚簇的垂直混合划分. 定义中 $T=\{t_1,t_2,\cdots,t_p\}$为任务集合.

当映射 ψ 定义为 Hash 映射时,对应的划分方式就叫作基于边聚簇的垂直混合 Hash 划分,这也是 BSP-VHP 系统采用的数据划分方法.

由于垂直混合划分对聚簇边进行重新分配,所以 BSP-VHP 需要一个新的路由信息来指明维护聚簇边的目的任务分区,因此下面提出聚簇边路由的概念.

定义 3. 聚簇边路由(clustered edges router, CER). 设映射 $\delta:\Phi\rightarrow T$ 表示聚簇边到对应任务的分配,划分给任务 j 的数据为 G'_j,对于顶点 $v\in G'$ 的聚簇边划分 $CER\leftarrow\{t_i\mid t_i\in\delta(\Phi(N_v,\psi)),\ i\neq j\}$,令 $G'_j\leftarrow(G'_j-CEL_i)\cup\{CER_i\}$,则称 CER_i 为顶点 v 到任务分区 t_i 的聚簇边路由.

图 3 说明了 BC-BSP 系统中采用的随机 Hash 划分方式,其等价于垂直混合划分时,约束条件 χ 为:采用取模 2 为 Hash 函数,且划分阈值 θ 为 ∞. 从图 3 可以发现,不存在满足垂直混合划分条件的聚簇边. 最大的消息通信次数为 7,没有对消息通信进行压缩处理.

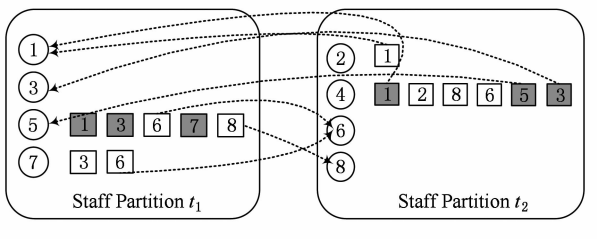


Fig. 3 Vertical Hash partitioning.

图 3 水平 Hash 划分

如果划分阈值 $\theta=3$,则任务分区 t_1 中顶点 5 的聚簇边划分为$\{CEL_{1,5}=\{1,3,7\}\}$,t_2 中顶点 4 的聚簇边划分为$\{CEL_{1,4}=\{1,3,5\},CEL_{2,4}=\{2,6,8\}\}$,其余所有边保持不变. 图 4 描述了具体的划分和放置情况,G'存放经过垂直切分压缩后的边路由及聚簇边路由信息,G''存放垂直切分后的聚簇边列表,虚线部分表示垂直划分的本地聚簇边,其对通信无影响. 数据划分后单次迭代的通信减少了 2 次.

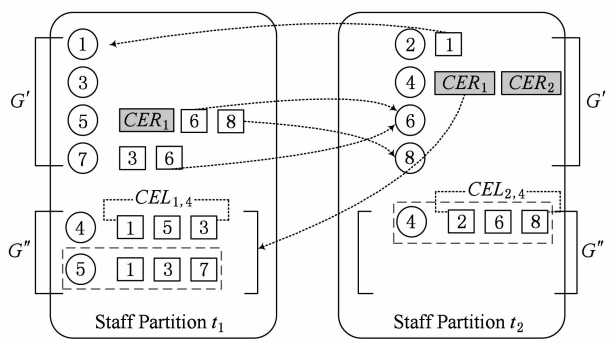


Fig. 4 Edge cluster based vertically hybrid partitioning.

图 4 基于边聚簇的垂直混合划分

在数据加载的过程中,部分处理任务的数据加载器从 HDFS 中读取一条记录 r,根据约束生成

聚簇边列表.根据顶点垂直划分阈值 θ 确定聚簇边列表中的可划分元素,并将该聚簇边划分至对应的任务数据分区的 G'' 中,更新路由 G'.具体计算过程如算法 1 所示.

算法 1. 垂直划分算法 $VPartition(Rr,\theta)$.

输入:数据加载器句柄 Rr、划分阈值 θ;

输出:边路由 G'、边计算分区 G''.

① Record $r=next(Rr)$; /* 获取下一顶点记录 */

② While $r\neq\varnothing$ Do {

③ HashMap $T=splitByPid(r)$;/* 聚簇边解析 */

④ List $L=\{\}$,$R=\{\}$;

⑤ For each partition p in T Do { /* 对聚簇边处理 */

⑥ If $size(p)\geqslant\theta$ then /* 满足划分阈值 */

⑦ Add p into R; /* 将聚簇边加入候选集合 */

⑧ Add $getPid(r)$ into L; /* 增聚簇边路由 */

⑨ Else /* 不满足划分阈值 */

⑩ Add p into L; } /* 聚簇边还原 */

⑪ If $getPid(r)=local$ then /* 是本地任务数据 */

⑫ add L to G'; /* 图顶点记录划分至本地 */

⑬ Else /* 非本地任务数据 */

⑭ $send(L,getPid(r))$; } /* 顶点记录分至其他任务 */

⑮ For each partition p in R Do {

⑯ If $getPid(p)=local$ then

⑰ continue; /* 本地聚簇边不划分 */

⑱ Else

⑲ $send(p,getPid(p))$; /* 聚簇边划分至其他任务 */

⑳ $r=next(Rr)$;

㉑ Return G',G''. }

下面讨论垂直混合划分策略对通信的影响.

设图 G 有 $|E|$ 条边、$|V|$ 个顶点,$|V|$ 个记录均匀分布在 p 个任务上.假设一个顶点 v 向其出度边点发送消息,计算网络上的消息规模.

对于任一顶点 v,其平均出度为 $|E|/|V|$,由于采用随机的均匀划分,除与 v 在同一个分区的出边顶点外,v 将会向其余任务分区发送 $(|E|/|V|)(p-1)/p$ 条消息,共有 $|V|$ 个顶点,则网络中的消息数为 $|E|(p-1)/p$.

因此,在未采用垂直混合划分的系统中,假设图 G 的顶点集合 V 均匀地划分在 p 个任务上,网络中产生的消息数量为式(1)所示:

$$Mssg_{\text{BC-BSP}}=|E|\cdot\frac{(p-1)}{p},\tag{1}$$

而在 BSP-VHP 系统中,v 原本向分区 p_i 发送的若干消息,现在压缩到只发送 1 条,那么设每个顶点向外发送 M_i 条消息,其上限为 $(p-1)$,一共有 $|V|$ 个顶点,则网络中产生的消息数上限为 $|V|(p-1)$.

因此,在基于垂直混合划分的 BSP-VHP 系统中,将图 G 的 $|V|$ 个顶点随机划分在 p 个任务上,则完全垂直划分后网络中产生的消息数量为式(2)所示:

$$Mssg_{\text{BSP-VHP}}=\sum_{i=1}^{n}M_i\leqslant|V|(p-1).\tag{2}$$

由式(1)和式(2)可以得出,BSP-VHP 的通信量与 BC-BSP 的通信量之比为式(3)所示:

$$\frac{Mssg_{\text{BSP-VHP}}}{Mssg_{\text{BC-BSP}}}=\frac{\sum_{i=1}^{|V|}M_i}{|E|\cdot\frac{(p-1)}{p}}\leqslant\frac{|V|}{|E|}\cdot P.\tag{3}$$

式(3)表明,使用基于边聚簇的垂直混合划分策略将平均出度为 $|E|/|V|$ 的图划分到 p 个任务上,一般情况下,若平均出度大于 p,则 $Mssg_{\text{BSP-VHP}}/Mssg_{\text{BC-BSP}}$ 的值也小于 1,这样通信量会明显降低,该比值越小通信开销降低越明显.而对于平均出度小于任务数 p 的情况,一般只采用水平的划分方式.

2.2 计算模型

为了降低 BC-BSP 系统以顶点为中心的计算处理负担,本节在 2.1 节的基于边聚簇的垂直混合划分策略的基础上提出了点-边计算模型.其处理流程如图 5 所示:

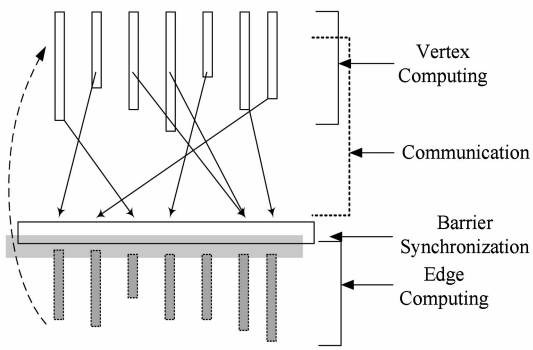


Fig. 5 Vertex-edge computing model.

图 5 点-边计算模型

在基于点-边模型的计算中,BSP-VHP 的计算引擎主要完成两部分计算:首先进行基本的以点为中心的本地计算,利用垂直混合划分后的数据产生边消息和聚簇边消息,通过异步的消息收发机制进行消息传递并通过路障同步保证两种消息全部发送到目的端的消息接收队列;然后边计算引擎对接收到的聚簇边消息进行处理,生成本地点计算需要的边消息.完成上述过程后系统进入下一次迭代的点计算处理阶段.

为了实现上述计算流程,用户必须重写该模型的两个接口函数 *VCompute*()与 *ECompute*()来实现具体的应用计算逻辑.图 6 显示出点-边模型下单源最短路径算法的编程实现过程:

```
Class ShortestPathBSP extends BSP {
  void VCompute(Iterator<Message> msgs, StaffContex cxt){
    int minDist=cxt.isSource()?0: INF;
    while(msgs.hasNext())
      minDis=min(minDist, msgs.next().value());
    if(minDist<cxt.getValue()){
      cxt.updateVertexValue(minDist);
      Iterator<Edge>edgeItr=cxt.getEdgeIterator();
      while(edgeItr.hasNext()){
        Edge e=edgeItr.next();
        cxt.sendMsgs(e, minDist+e.weight());
      }
      Iterator<EdgePartition> edgePart=cxt.getEdgePartition
        ();
      while(edgePart.hasNext())
      cxt.sendPartitionMsgs(edgePart.next(), minDist);
    }
  cxt.vertexToHalt();
  }
  void ECompute(Message msg, StaffContext cxt){
    Iterator<Edge> edgeItr=cxt.getPartitionEdgeIter();
    while(edgeItr.hasNext()){
      Edge e=edgeItr.next();
      cxt.incomeMsg(e, msg.value()+e.weight());
    }
  }
}
```

Fig. 6 The example of SSSP.

图 6 点-边计算模型下单源最短路径实现

在该算法中,假设每个顶点的最短路径值的初始值为无限大.在每一次迭代计算时,点计算中每个顶点首先接收来自其入度边方向的相邻顶点发来的消息,并选择最小值作为该顶点更新的备选值,如果这个最小值小于当前该点最短路径值,则更新该顶点的最短路径值,同时向该顶点的出度相邻顶点发送边消息与聚簇边消息.这些邻居顶点在接收到消息后,若是边消息则直接加入边消息接收队列,等待下一次迭代中的点计算处理;若接收到的是聚簇边消息则在点计算之前进行边计算,再将边计算生成的边消息加入到下一次迭代的消息处理队列中.

2.3 代价收益模型与阈值 θ 分析

根据 2.1 节的内容,参数 θ 的取值越小垂直混合划分粒度越细,通信代价降低得越多,但是系统整体运行效率的提高与 θ 取值并非简单的反比关系,因为垂直混合划分策略的实现相比于 BSP-VHP 中原有的简单 Hash 划分,在数据加载时增加了 CPU 的计算代价(聚簇边解析)和网络通信开销(聚簇边重新分配).此外,在迭代计算过程中,点-边计算模型的边计算需要聚簇边消息与目的分区的聚簇边之间的匹配查找,这会引入额外的 CPU 开销.例如当 $\theta=1$ 时,所有数据均需要被划分,建立聚簇边路由以及在目的分区进行边计算.同时,当 $\theta=\infty$ 时,划分策略退化为基本的随机 Hash 划分,处理相同图数据集时消息规模与 BC-BSP 相同.因此,θ 的取值与垂直混合划分对系统性能的影响关系紧密,因而本节将对垂直混合划分的代价和收益进行讨论,分析 θ 的最优取值.

设在任意一个处理机的任务处理单元 t_0 中,对所有满足划分阈值 θ 的聚簇边按照规模进行排序分组,可以得到垂直划分直方图,简记为 ζ.

定义 4. 垂直划分直方图.对于划分参数 θ 的任一取值,可划分的聚簇边的数目 $|\zeta|_{\text{total}}$ 为

$$|\zeta|_{\text{total}}=\sum_{i\geqslant\theta}\zeta_i, \tag{4}$$

其中,$\zeta_i=\dfrac{\sum|E_k|}{i}$, $\forall v\in|V|$, $\forall E_k\in\Phi$, $E_k\subseteq N_v$,且 $|E_k|=i$.ζ_i 就叫作垂直划分的第 i 级直方图.

基于垂直划分直方图 ζ,对于指定的划分参数 θ,可以确定网络通信的减少量 $Mssg_{\text{reduced}}$,其等于边路由消息的减少量 $Mssg_{\text{cut}}$ 减去聚簇边路由消息的增加量 $Mssg_{\text{inc}}$,因此有,

$$Mssg_{\text{cut}}=\sum_{i\geqslant\theta}i\zeta_i; \tag{5}$$

$$Mssg_{\text{inc}}=\sum_{i\geqslant\theta}\zeta_i; \tag{6}$$

$$Mssg_{\text{reduced}}=\sum_{i\geqslant\theta}(i-1)\zeta_i. \tag{7}$$

总体代价 *Cost* 主要包括数据的加载代价 $Cost_{\text{load}}$ 和对 G'' 结构的查找代价 $Cost_{\text{search}}$.数据加载代价主要为划分算法执行的 CPU 代价 $Cost_{\text{loadCPU}}$ 和划分聚簇边的通信代价 $Cost_{\text{loadComm}}$,而:

$$Cost_{\text{loadCPU}} = \frac{|V| c(m)}{q}, \quad (8)$$

其中,c 为平均出度为 m 时的单位处理代价,$m = |E|/|V|$.

$$Cost_{\text{loadComm}} = \frac{c_1(n)}{q} \cdot \sum_{i \geqslant \theta} \zeta_i, \quad (9)$$

其中,q 为作业的数据加载器数,c_1 为聚簇边平均出度为 n 时单位通信开销. 所以有:

$$Cost_{\text{load}} = \frac{|V| c(m)}{q} + \frac{c_1(n)}{q} \cdot \sum_{i \geqslant \theta} \zeta_i. \quad (10)$$

而聚簇边计算的查找开销为

$$Cost_{\text{search}} = s \cdot \frac{c_3}{p} \cdot \sum_{i \geqslant \theta} \zeta_i, \quad (11)$$

其中,p 为作业的任务数,c_3 为单位查找代价,s 为迭代次数,则总代价 $Cost$ 为

$$Cost = \frac{|V| c(m)}{q} + \frac{c_1(n)}{q} \cdot \sum_{i \geqslant \theta} \zeta_i + s \cdot \frac{c_3}{p} \cdot \sum_{i \geqslant \theta} \zeta_i. \quad (12)$$

单次迭代的通信收益为

$$Benefit = \frac{c_0}{p} \cdot \sum_{i \geqslant \theta} (i-1)\zeta_i, \quad (13)$$

其中,c_0 表示单个消息的传输代价,所以 BSP-VHP 系统进行 s 步迭代的性能总增益为

$$Gain = Benefit - Cost = s \cdot \frac{c_0}{p} \cdot \sum_{i \geqslant \theta} (i-1)\zeta_i - \frac{|V| c(m)}{q} - \frac{c_1(n)}{q} \cdot \sum_{i \geqslant \theta} \zeta_i - s \cdot \frac{c_3}{p} \cdot \sum_{i \geqslant \theta} \zeta_i. \quad (14)$$

如果指定 θ 的取值,只要消息的发送收益大于聚簇边的查找代价就能获得正的总增益,通过去掉式(14)中常数项而化简式(14),有:

$$\partial = s \cdot \frac{c_0}{p} \cdot \sum_{i \geqslant \theta} \zeta_i (i-1) - s \cdot \frac{c_3}{p} \cdot \sum_{i \geqslant \theta} \zeta_i. \quad (15)$$

只需式(15)大于 0 就能保证存在迭代次数的下限阈值使得总增益为正. 展开求和公式后的通项公式为

$$\partial_i = \zeta_i \cdot \frac{s}{p}(i \cdot c_0 - c_0 - c_3). \quad (16)$$

通项公式的符号由括号内式子决定,且在 i 的值从正无穷到 1 的变化过程中,存在 i_0,使得满足式(17):

$$i_0 c_0 - c_0 - c_3 \leqslant 0 \leqslant i_0 c_0 - c_3, \quad (17)$$

其中,i_0 的实际取值与网络传输代价以及聚簇边的查找代价具体相关. 一般情况下网络传输的单位代价远远大于内存查找的单位代价,所以通常 i 取值为 2 时就会获得正的总增益.

3 并行点-边计算模型与索引机制

本文第 2 节提出的基于边聚簇的垂直混合划分策略以及点-边计算模型,本质上也可以看作是对计算负载的迁移,即将原来的本地计算任务的部分计算负载迁移到消息接收过程中进行处理. 而在消息的接收过程中系统对聚簇边消息的计算处理可以分为两种方式:串行的集中式计算与并行的即时计算.

串行集中式的计算模式如图 5 所示,这一计算过程需要等待点计算阶段完成,且消息通信同步结束后才能进入边计算阶段,等价于在每次的迭代过程中增加了一次对本地聚簇边的扫描计算,称之为串行扫描(serial scan, SS)计算. 因此,在 SS 计算模式下,每个任务的点计算与边计算是串行执行的,但其通信处理比较简单,只需要将聚簇边消息加入到相应的处理队列即可.

如图 7 所示,并行的即时计算指的是计算任务的聚簇边消息接收线程在接收到邻接顶点发来的聚簇边消息后,马上对该消息进行边计算,并将产生的边消息加入到相应的消息接收处理队列. 由于 BSP-VHP 系统采用的是异步并行的消息发送机制,所以即时计算的并行化是依赖于接收线程的,不同的源分区发来的聚簇边消息在不同的接收线程里被并行地即时处理. 可以认为,聚簇边消息的并行即时处理是对本地计算部分负载的并行化迁移,但同时这一过程增加了消息处理机制的负担并可能造成通信性能下降.

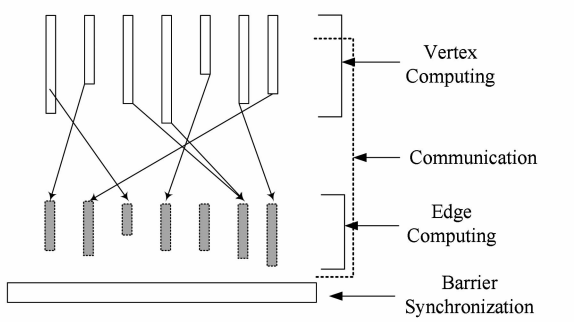


Fig. 7 The parallel Vertex-Edge computing model.

图 7 并行点-边计算模型

并行的即时计算过程是以消息为中心进行的,需要根据消息的源 ID 查找对应的聚簇边进行边计算. 这种计算的可行性基于聚簇边消息与聚簇边是一一对应的,而不会如点计算那样一个顶点接收到多条邻接顶点发来的边消息.

为了提高即时计算的效率,需要在数据划分时对聚簇边建立查询索引. 如图 8 所示,最简单的方式是建立 Hash 索引,在查找聚簇边消息时系统根据消息的源 ID 的 Hash 值查找聚簇边的 Hash 索引表,查找的时间复杂度为常数级,称这种索引为并行 Hash 索引(parallel Hash index, PHI). 但 PHI 的查找方式也增加了数据混合划分时的处理代价,数据的加载时间会增加. 与此同时,虽然 Hash 查找的理论时间是常数级的,但是随着处理数据的规模增大以及内存容量的限制,Hash 冲突加剧,最终导致 BSP-VHP 系统在实际的计算过程中的平均查找效率会降低.

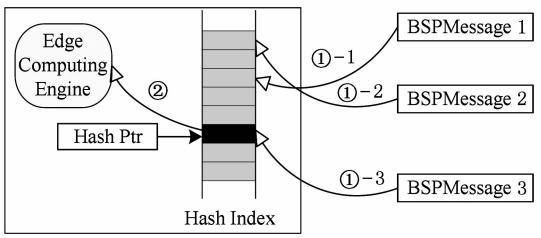


Fig. 8 Searching with Hash index.

图 8 Hash 索引查找

针对上面的问题,本文提出一种基于聚簇边有序的多队列并行顺序索引的机制,称之为并行顺序索引(parallel sequence index, PSI),其具体处理流程如图 9 所示. 计算任务在接收到来自其他任务分区发送来的聚簇边消息后,在本任务的接收线程中,根据聚簇边消息的 PID 值在 PSI 结构中找到对应的聚簇边分区队列,然后从该队列中获得当前的聚簇边指针(黑色矩形小块)所指的聚簇边元素,同时将队列指针移至下一位置. 将获得的聚簇边 ID 与消息中的 ID 相匹配,如果不匹配则获取下一聚簇边,直至匹配成功,然后请求边计算引擎进行处理,至此该聚簇边消息处理完毕.

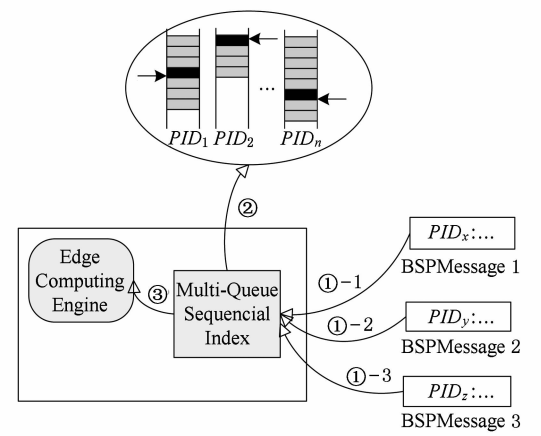


Fig. 9 Searching with parallel sequential index.

图 9 并行顺序索引查找

要保证 PSI 的有效性,关键是必须确保在数据划分时各个任务进行聚簇边重划分的顺序与迭代计算时边计算的顺序互相一致. 因此,基于聚簇边的垂直混合划分需要分两个步骤进行:首先,各个加载任务将图数据从 HDFS 上加载到本地,并按照基本的随机 Hash 划分策略进行数据的水平划分;然后,在所有处理任务得到对应的数据分区后,以计算任务为单位对初始划分后的数据分区再进行垂直混合划分,将满足划分阈值约束的聚簇边划分至对应任务的数据分区中.

从实验可以看出,并行即时计算的系统性能是优于串行的集中式计算的,并且多队列并行的顺序索引结构 PSI 的查找效率优于单队列的 Hash 索引 PHI. 实验中若无特别说明,BSP-VHP 系统采用多队列并行顺序索引机制.

4 实验结果与分析

本节通过在真实数据集与合成数据集上进行 PageRank 计算,对 BSP-VHP 系统中的数据划分策略进行实验评估,并将该系统的处理性能与 BC-BSP 和 Giraph 的处理性能进行对比. 真实数据集特征如表 1 所示:

Table 1 Description of Real Datasets

表 1 真实数据集描述

Dataset	Vertex Num	Edge Num	Size/MB
Wiki-talk	2 394 385	5 021 410	45.4
Live-J	4 847 571	68 993 773	700
Wiki-pp	5 716 808	130 160 393	1 500

实验环境是由 20 台计算节点组成的计算集群. 其中每个计算节点的配置为:Intel Core i3-2100 双核处理器;8 GB RAM (每个任务默认分配 2GB);7200 RPM 硬盘;硬盘容量为 500 GB. 每个计算节点的最大任务槽数设为 2.

1) 不同的点-边计算模式对系统性能的影响:图 10 和图 11 分别给出了串行扫描 SS、并行 Hash 索引 PHI 与多队列并行顺序索引 PSI 这 3 种计算模式对 BSP-VHP 系统的处理性能与数据加载时间的影响.

从图 10 可以看出,在 θ 取值为 2,4,7 时,3 种计算模式均具有较好的处理性能,且并行顺序索引的

性能最好,并行 Hash 索引次之,顺序扫描最差. 这是因为 PHI 与 PSI 对边计算任务进行了部分的负载并行化迁移,提高了系统的分布式计算效率,且在两种并行的即时计算方法中 PSI 的查找速度快于 PHI.

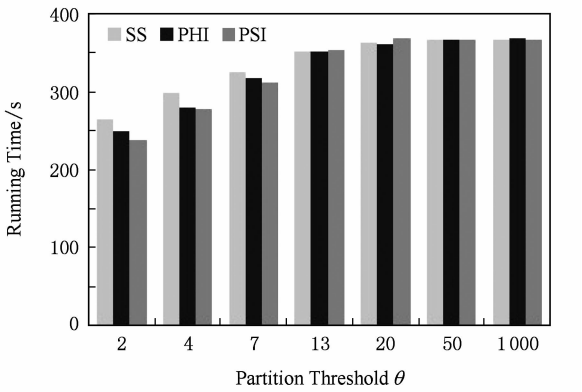


Fig. 10 Comparison of running time at different Vertex-Edge computing mode(Wiki-pp).

图 10 不同点-边计算模式下运行时间的比较(Wiki-pp)

图 11 揭示了 3 种计算模式对数据加载时间的影响. 可以看出,当 θ 取值较小时,因划分时满足条件聚簇边数量较多,3 种计算模式的数据加载时间均增加. 但是因为 PSI 需要进行 2 次数据划分,所以数据加载的时间代价最高. 由于 PHI 在基本的数据混合划分过程中增加了建立 Hash 表的操作,所以代价高于 SS 的数据加载代价.

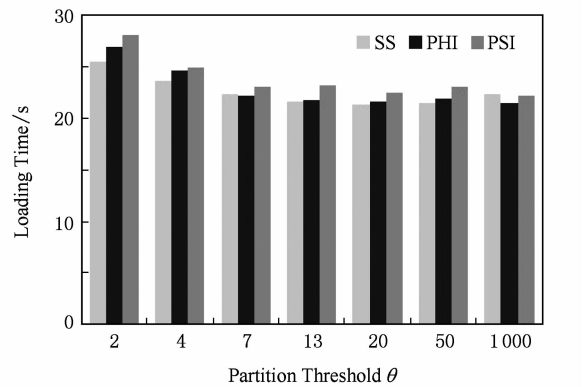


Fig. 11 Comparison of loading time at different Vertex-Edge computing mode(Wiki-pp).

图 11 不同点-边计算模式下加载时间的比较(Wiki-pp)

2) 划分参数 θ 的影响. 图 12 和图 13 分别给出了划分阈值 θ 的变化对于消息规模与系统运行总时间的影响.

由图 12 可以看出,随着划分阈值 θ 从 1 逐渐增大,BSP-VHP 系统中的消息规模不断增大. 这是因为划分阈值 θ 越小,满足划分条件的聚簇边越多,越多的聚簇边路由替换原来的边路由,通信规模的压缩程度越大. 从实验结果可以看出:EC-VHP 在不同数据集上产生的作用效果并不相同,其中在 Wiki-pp 数据集上消息通信量减少了约 2/3,而在 Wiki-talk 数据集中基本没有效果.

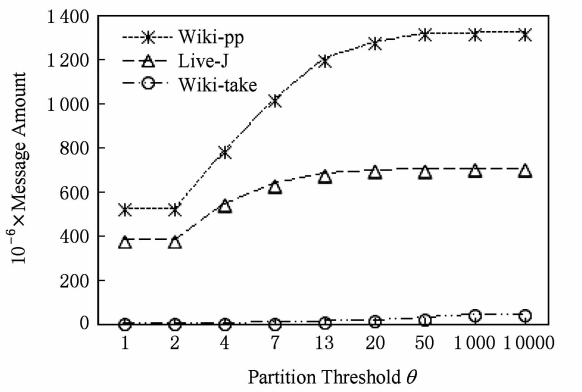


Fig. 12 Variation of message scale as a function of partitioning threshold θ.

图 12 消息规模随划分阈值 θ 的变化图

图 13 反映了运行时间随划分阈值 θ 的变化情况. 从图中可以看出,运行时间的变化与图 12 通信消息量的变化趋于一致. 系统对 Wiki-pp 数据集的处理时间变化最大,在划分阈值为 2 时,BSP-VHP 系统几乎减少了 1/3 的处理时间;而对于数据集 Live-J,在划分阈值较小时运行时间也得到了减少,但是相对而言性能提高不多;对于 Wiki-talk 数据集,运行时间几乎没有变化. 这是因为 BSP-VHP 系统在处理数据集 Wiki-pp 时,小划分阈值导致消息通信规模的大幅度降低,而在处理后两个数据集时消息规模的变化幅度较小,产生的性能提高效果不明显. 实际上,当 EC-VHP 作用于不同的数据集时,运行结果会受数据集自身的特性(主要是平均出度)的影响. 后续实验将进一步验证该观点.

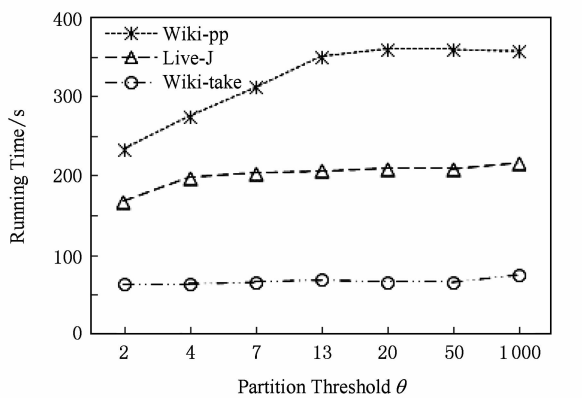


Fig. 13 Variation of running time as a function of partitioning threshold θ.

图 13 运行时间随划分阈值 θ 的变化图

3) 图处理系统间的性能对比. 图 14 反映的是 BSP-VHP,BC-BSP 和 Giraph(0.1 版本)3 个系统在真实数据集下的性能对比. 可以发现,BSP-VHP 系统在 Live-J 与 Wiki-pp 数据集上有较好的表现,在对后者进行 PageRank 计算时,BSP-VHP 是 Giraph 的 3 倍,是 BC-BSP 系统的 1.4 倍.

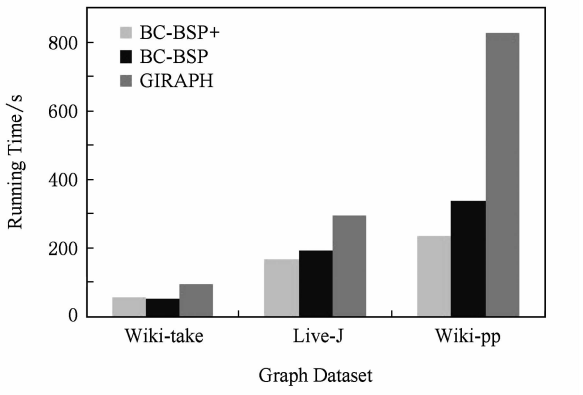


Fig. 14 Comparison of performance on three systems at different datasets($\theta=2$).

图 14 不同数据集上 3 个系统的性能对比($\theta=2$)

4) 图数据结构对垂直混合划分的影响. 图 15 与图 16 分别反映了图数据边的规模与点的规模对本文提出的划分方法的影响.

在图 15 中,为了验证图数据的平均初度(或图的稠密程度)对于 BSP-VHP 的影响,采用控制图顶点规模保持固定值而增加平均出度(增量地随机增加边的数目)的方式,其中,图顶点的规模固定为 100 万. 从图 15 可以发现,随着图数据平均出度的增加,采用不同的划分阈值对 BSP-VHP 运行时间的影响也不同. 当图数据平均出度为 100 时,$\theta=2$ 的处理速度比 θ 取较大数值时的处理速度快将近 1 倍. 所以垂直混合划分方法对于图数据的平均出度变化是敏感的.

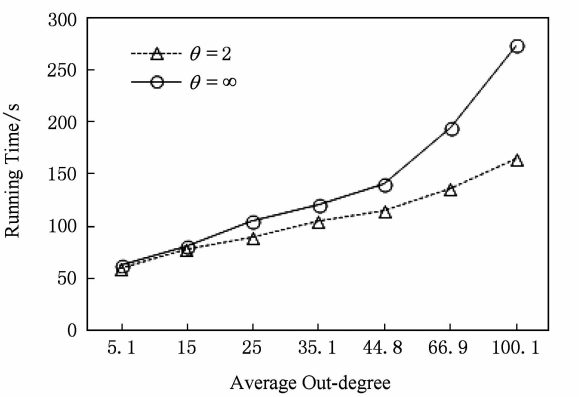


Fig. 15 Variation of running time as a function of average out-degree.

图 15 系统运行时间随平均出度的变化

图 16 反映了当图数据平均出度相同时,图数据规模在不同划分粒度下对系统性能的影响. 在该实验中,采用控制图的平均出度(或稠密程度)保持不变而增大图顶点规模的方式进行验证. 从图 16 可以看出,随着图顶点的规模从 100 万增加至 500 万,对系统运行时间没有明显的影响,所以本文的数据划分方法对图数据的规模变化是不敏感的.

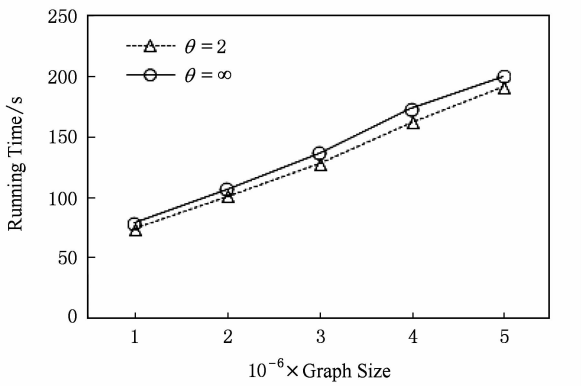


Fig. 16 Variation of running time as a function of graph size.

图 16 系统运行时间随数据规模的变化图

5 结 论

本文针对基于 BSP 模型的图处理系统中普遍存在的消息通信性能瓶颈问题,首先从数据划分的角度提出了基于边聚簇的垂直混合划分策略,并基于该策略设计了点-边计算模型,然后就划分阈值的选取与垂直混合划分对系统性能的影响进行了分析,对于绝大多数情况,划分阈值为 2 时系统获得最佳处理性能. 最后,本文在点-边计算模型基础上提出了并行的点-边计算模型,并设计实现了能够提高系统计算并行度的两种索引机制,即并行 Hash 索引与多队列并行顺序索引. 实验结果表明,本文提出的基于边聚簇的垂直混合划分策略能够有效地提高 BSP-VHP 系统的计算效率,而并行的点-边计算模型与两种索引结构进一步增强了系统的并行化程度. 由实验可以看出本文提出的数据划分策略和设计的索引机制对于处理高出度图数据尤为有效. 本文提出的垂直混合划分策略与水平划分方式无关,而 BSP-VHP 系统目前仅针对 Hash 划分方式实现了垂直混合划分,但并没有考虑基于其他水平划分方式的实现. 本文通过点-边计算模型减轻点计算的处理负担,但并没有进一步对大图数据中因顶点出度

边偏斜而造成的负载不均衡问题予以解决.未来工作将围绕上述内容开展.

参 考 文 献

[1] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113

[2] White T. Hadoop: The Definitive Guide [M]. Translated by Zhou Minqi, Wang Xiaoling, Jin Cheqing, et al. 2nd ed. Beijing: Tsinghua University Press, 2011 (in Chinese)
(怀特汤姆. Hadoop 权威指南. 周敏奇, 王晓玲, 金澈清, 等译. 2 版. 北京: 清华大学出版社, 2011)

[3] Pace M F. Hama vs MapReduce [EB/OL]. [2012-06-25]. http://arxiv.org/abs/1203.2081

[4] 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: USENIX Association, 2012: 31-46

[5] Grzegorz M, Austern H M, et al. Pregel: A system for large-scale graph processing [C] //Proc of the ACM SIGMOD 2010 Int Conf on Management of Data. New York: ACM, 2010: 135-146

[6] Chen Ruishan, Weng Xuetian, He Bingshen, et al. Large graph processing in the cloud [C] //Proc of 2010 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2010: 1123-1126

[7] Sangwon S, Edward J Y. HAMA: An efficient matrix computation with the MapReduce framework [C] //Proc of the 2nd IEEE Int Conf on Cloud Computing Technology and Science. Los Alamitos, CA: IEEE Computer Society, 2010: 721-726

[8] The Apache Software Foundation. Introduction to Giraph [EB/OL]. [2013-06-25]. http://giraph.apache.org/intro.html

[9] Feng Guodong, Xiao Yanghua. The distributed storage of big scale graph [J]. Communications of the CCF, 2012, 8(11): 11-8 (in Chinese)
(冯国栋, 肖仰华. 大图的分布式存储[J]. 中国计算机学会通讯, 2012, 8(11): 11-8)

[10] Catalyurek U V, Aykanat C. Patoh: Partitioning tool for hypergraphs [CP/OL]. [2013-06-25]. http://bmi.osu.edu/~umit/software.html

[11] Hendrickson B, Leland R. An improved spectral graph partitioning algorithm for mapping parallel computations [J]. SIAM Journal on Scientific Computing, 1995, 16(2): 452-469

[12] Pujol J M, Erramilli V, Rodriguez P. Divide and conquer: Partitioning online social networks [EB/OL]. New York: Cornell University Library [2013-06-26]. http://arxiv.org/pdf/0905.4918v1.pdf

[13] Blondel V D, Guillaume J L, Lefebvre E, et al. Fast unfolding of communities in large networks [J/OL]. Journal of Statistical Mechanics: Theory and Experiment, 2008 [2013-06-25]. http://iopscience.iop.org/1742-5468/2008/10/P10008

[14] Leskovec J, Dasgupta A, Mahoney M W. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters [J]. Journal of Internet Mathematics, 2009, 6(1): 29-123

[15] Newman M. Why social networks are different from other types of networks [J]. Physical Review E, 2003, 68(3): 1-9

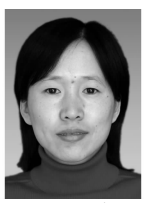
[16] Mondal J, Deshpande A. Managing large dynamic graphs efficiently [C] //Proc of 2012 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2012: 145-156

[17] Salihoglu S, Widom J. GPS: A graph processing system [C] //Proc of the 25th Int Conf on Scientific and Statistical Database Management. New York: ACM, 2013

[18] Gonzalez J E, Low Y. PowerGraph: Distributed graph-parallel computation on natural graphs [C] //Proc of the 10th USENIX Conf on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2012: 17-30

[19] Bao Yubin, Wang Zhigang, Yu Ge, et al. BC-BSP: A BSP-based parallel iterative processing system for big data on cloud [C] //Proc of the 1st Int DASFAA Workshop on Big Data Management and Analytics. Berlin: Springer, 2013: 31-45

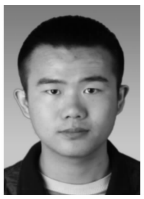
[20] Khayyat Z, Awara K, Alonazi K, et al. Mizan: A system for dynamic load balancing in large-scale graph processing [C] //Proc of the 8th ACM European Conf on Computer Systems. New York: ACM, 2013: 169-182



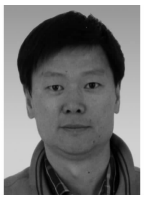
Leng Fangling, born in 1978. Received her PhD degree in computer software and theory from Northeastern University in 2008. Lecturer at Northeastern University. Member of China Computer Federation. Her research interests include data warehouse and online analytical processing (OLAP), etc.

Liu Jinpeng, born in 1989. Received his master degree at the College of Information Science and Engineering, Northeastern University. His main research interests include cloud computing and graph management, etc.

Wang Zhigang, born in 1987. PhD candidate at the College of Information Science and Engineering, Northeastern University. His research interests include cloud computing and graph data mining, etc.



Chen Changning, born in 1990. Received his master degree at the College of Information Science and Engineering, Northeastern University. His research interests include cloud computing and graph data mining, etc.



Bao Yubin, born in 1968. Received his PhD degree in computer software and theory from Northeastern University in 2003. Professor at Northeastern University. Senior member of China Computer Federation. His research interests include data warehouse, online analytical processing (OLAP), cloud computing and data intensive computing, etc.

Yu Ge, born in 1962. Received his PhD degree in computer science from Kyushu University of Japan in 1996. Professor and PhD supervisor at Northeastern University. His research interests include database theory and techology, distributed system, parallel computing and cloud computing, etc.

Deng Chao, born in 1978. Received his PhD degree in computer science from Harbin Institute of Technology in 2009. Research professor at China Mobile Research Institute. His research interests include big data analysis and mining, distributed parallel computing, etc.