周　爽,鲍玉斌,王志刚,等.BHP:面向BSP模型的负载均衡Hash图数据划分[J].计算机科学与探索,2014,8(1):40-50.

ISSN 1673-9418　CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2014/08(01)-0040-11
doi: 10.3778/j.issn.1673-9418.1305052

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-89056056

BHP:面向BSP模型的负载均衡Hash图数据划分*

周　爽[1],鲍玉斌[1+],王志刚[1],冷芳玲[1],于　戈[1],邓　超[2],郭磊涛[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

BHP:BSP Model Oriented Hash Graph Data Partition with Load Balancing*

ZHOU Shuang[1], BAO Yubin[1+], WANG Zhigang[1], LENG Fangling[1], YU Ge[1], DENG Chao[2], GUO Leitao[2]
1. College of Information Science and Engineering, Northeastern University, Shenyang 110819, China
2. Division for Business Support, China Mobile Institute, Beijing 100053, China
+ Corresponding author: E-mail: baoyubin@ise.neu.edu.cn

ZHOU Shuang, BAO Yubin, WANG Zhigang, et al. BHP: BSP model oriented Hash graph data partition with load balancing. Journal of Frontiers of Computer Science and Technology, 2014, 8(1): 40-50.

Abstract: Graph data partition is a critical technical problem in the large-scale graph processing system based on bulk synchronous parallel (BSP) model. Traditional graph partition technology requires many iterations, the time complexity is too high, and the partition results don't have the mapping information from vertexes to partitions. So those algorithms are not suitable for partitioning graph data for the BSP model based systems. This paper presents a new Hash partition algorithm that implements load balance, called balanced Hash partition (BHP). In order to achieve the balance of the number of out degrees in each partition as much as possible, the concept of virtual bucket is introduced. Then these virtual buckets are reorganized into desired partitions by using a greedy algorithm. In this way, the algorithm can ensure the load balance of each partition. At the same time, the data localization strategy makes the data on the split locate on the corresponding node as much as possible. So, the overhead of data migration during the data loading can be reduced. This paper does some experiments to compare the performance between BHP and the classic Hash algorithm from multiple aspects. The results show that the BHP algorithm can improve the job executing efficiency,

* The National Natural Science Foundation of China under Grant Nos. 61033007, 61173028 (国家自然科学基金); the Fundamental Research Funds for the Central Universities of China under Grant No. N100704001 (中央高校基本科研业务费专项资金); the Ministry of Education of China and China Mobile Foundation under Grant No. MCM20125021 (教育部-中国移动科研基金).
Received 2013-04, Accepted 2013-06.
CNKI网络优先出版:2013-07-18, http://www.cnki.net/kcms/detail/11.5602.TP.20130718.1603.003.html

reduce the number of sending messages, and effectively solve the problems of load imbalance and too many interactive edges among partitions.

Key words: bulk synchronous parallel (BSP); graph partition; distributed system; load balance; virtual bucket

摘　要:图数据划分是基于BSP(bulk synchronous parallel)编程模型的大规模图处理系统中一个关键技术问题。传统的图划分技术需要多次迭代,时间复杂度过高,且划分结果不具有图顶点到分区的映射信息,因此这些算法并不适用于BSP模型下的数据划分。提出了一种新的面向BSP模型的负载均衡Hash数据划分算法(balanced Hash partition,BHP)。为了实现各个分区的出边数尽可能均衡,该算法引入了虚拟桶的概念,通过贪婪算法将虚拟桶重组为实际分区,保证了每个实际分区负载均衡,同时数据本地化策略使本分片上的数据尽可能地保留在本节点上,从而减小在数据加载时的数据迁移开销。从三个方面对比了BHP算法和经典Hash算法的性能,结果表明BHP算法能够提高作业的执行效率,减少消息发送的数量,有效解决了经典Hash算法的负载不均衡和分区间交互边过多的问题,当数据量变大时,效果尤为明显。

关键词:BSP模型;图划分;分布式系统;负载均衡;虚拟桶

文献标志码:A　**中图分类号:**TP311

1　引言

近年来,BSP(bulk synchronous parallel)模型[1]被广泛用于大图数据的并行分布式处理中,它是一种基于消息通信的整体同步并行计算模型。BSP模型把并行计算抽象为三个模块:处理器集合、发送消息的全局通讯网络和各处理器间的路障同步机制。BSP模型中将一个超步(SuperStep)作为并行计算的基本执行单元。一个BSP程序包含多个超步,通过消息机制实现超步中各个计算任务间的信息传递。BSP模型迭代执行每一个超步,直到满足终止条件或者达到一定的超步数强制终止。在集群环境下进行大规模图数据的分布式并行处理的一个重要任务就是图数据的划分[2],即决定一个图顶点应该属于哪个子图或分区,并将各个图分区数据分配给各个处理器进行处理。因为图数据本身具有连通性,所以为了实现高效的图数据分布式并行处理,进行合理的图数据划分十分重要。图数据划分的原则有两点:一是降低划分后子图间的交互边,从而降低计算过程中发送的消息量;二是保证子图处理的均衡性,即并行计算中各处理任务的负载均衡问题。图数据划分时应尽可能考虑满足上述原则。

虽然有很多种经典的图划分算法,例如Kernighan-Lin算法[3-4]、METIS算法[5]、基于Laplace图特征值谱二分法[6-7]等,但是这些划分技术都需要多次迭代,时间复杂度过高,而且划分结果不保留图顶点到分区的映射信息,因此并不适用于BSP模型下的数据划分。而经典的Hash划分算法对于海量的,特别是顶点ID类型不确定(例如整型或字符串类型)的图数据往往会造成数据偏斜,或者划分时数据可能被完全散列到所有节点上,而没有考虑图顶点之间的关联性,使得分区间交互边过多。因此,如何快速实现图数据的划分是基于BSP模型大图数据处理的难点,具有极大的挑战性。

基于上述问题,本文提出了一种适合于BSP模型的负载均衡Hash图数据划分算法(balanced Hash partition,BHP),该算法在一定程度上实现了各个计算任务的负载均衡,有效地降低了水桶效应。本文从消息通信(即网络IO),集群中节点负载均衡,图数据加载用时,以及以PageRank[8]应用为例的计算耗时等方面,将本文提出的BHP算法与经典的Hash算法进行了实验对比,分析总结了BHP算法的优缺点。

本文组织结构如下:第2章简要介绍了相关工作,并回顾了几种经典的图划分算法;第3章分析了经典Hash划分算法及其优缺点;第4章提出了一种近似的BSP作业运行的时间代价评估模型;第5章详细描述了本文提出的负载均衡Hash图数据划分算法;第6

章给出了与经典Hash划分算法的对比实验;最后总结了全文的工作。

2 相关工作

由于社会媒体和Web搜索应用的出现,许多应用(例如网页的PageRank计算)需要处理海量的数据,而一个大搜索引擎一年内爬取下载的网页有近一万亿个顶点及十万亿个出边[9]。因此,对海量图数据处理的解决方案是将图数据集划分到计算机集群的各个节点上,然后采用分布式算法进行处理。这就需要考虑数据的分布及划分。这里图划分定义为将一个图 $G=(V,E)$ 作为输入,并将图 G 划分成 k 个子图 $G_1=(V_1,E_1), G_2=(V_2,E_2), \cdots, G_k=(V_k,E_k)$, 满足 $V_i \cap V_j = \Phi(i \neq j)$ 且 $\bigcup_{i=1}^{k} V_i = V$,最后将 k 个子图分配给集群中的 k 个任务。基于BSP模型的图处理系统中图划分的一个关键要求是图数据划分时耗时尽量小,划分后各个任务的负载尽可能均衡,而且划分后能够根据一个顶点的ID计算出其位于哪个分区上。

Pregel[10]、Hama[11]、Giraph[12]、GraphLab[13-14]和PowerGraph[15]是目前比较流行的基于BSP模型的大图处理系统。Pregel是由Google提出的用于分布式图计算的计算框架。Hama是Apache的Hadoop[16]中的子项目,被看做Pregel的开源实现。Giraph是Yahoo提出实现的,建立在Hadoop基础上的面向图处理的算法库。GraphLab和PowerGraph是面向机器学习的高性能的并行流处理框架。它们都采用了BSP模型,其中Pregel、Hama和GraphLab采用了经典的Hash取模静态划分算法。

Pregel、Hama和GraphLab中处理的图以邻接表的方式存储。每个顶点有一个ID作为唯一的标识,采用直接取余法,将一个顶点利用Hash(ID) mod p=k的方式把该顶点映射到第 k 个存储分区, p 代表分区的数目。这种划分不能保证每个分区中的图顶点数量均衡,亦不能保证分区内边数的均衡。

Giraph提出了一种动态的图划分算法。Giraph中主控节点以顶点分片(VertexSplit)的逻辑概念将图数据划分为多个分片。每个工作节点负责处理一个顶点分片。工作节点对自己负责的顶点分片进一步划分为一系列的顶点范围(VertexRange)对象。随后工作节点将划分后的信息上报给主控节点。在每个计算超步前,主控节点把这些顶点范围对象根据一定的策略分配给参与计算的机器并形成一张映射表。这样做的好处是处理粒度大,重划分以顶点范围为基本单元进行整体的迁移,易于实现重新定位。但是由于在每个超步前都要进行重划分,并不适用于迭代次数较少的作业。

PowerGraph的核心思想是对顶点进行切分。一个顶点可以被部署到多台机器,其中一台作为主控顶点,其余的作为镜像顶点。主控顶点是所有镜像顶点的管理者,镜像顶点与主控顶点保持数据一致。PowerGraph将每条边唯一地部署在某一节点上,从而避免了边的冗余。

在上述五个系统中,划分方法的使用场景与本文讨论算法的使用场景是一样的,这些算法都很简单,但是前四个没有考虑边的负载均衡问题,PowerGraph考虑了边的负载均衡问题,主要用于具有超大边表的图数据处理。

图作为一种经典的数据结构得到了广泛的研究,在图划分方面已经有一些经典的算法被提出。Kernighan-Lin算法是一种试探优化法,利用贪婪算法将复杂网络划分为两个子图的二分法。分割过程中既考虑每个子图的聚簇特性,同时又要保证分割后的子图在数据规模上的均衡性。不过K-L算法的时间复杂度较高,为 $O(n^2 \operatorname{lb} n)$,其中 n 是图顶点的数量。METIS是一个实现了很多算法的程序包,其中提供了一个图划分算法。METIS采用多级别图划分技术,将划分过程分为粗化阶段、划分阶段和反粗化三个阶段。由于划分过程分为三个阶段,且在划分阶段仍采用已有的图划分算法(如K-L算法)进行数据划分,它依旧保留了许多经典算法的缺点。基于Laplace图特征值谱二分法[17-18]也是一个经典的图划分算法,基本思想是找到Laplace矩阵中不为0的特征值,并取得对应的特征向量的线性组合。该方法只能对图进行二分,且时间复杂度较大,最坏情况下为 $O(n^3)$ 。GN算法[19]是一种分裂算法。通过不断地从网络中删掉介数(betweeness)最大的边从而实现划分。GN算法考虑了网络全局,划分得到的社区的

准确度较高。但是对网络社团结构缺少量的定义,需要事先知道社团的个数。这些经典的划分技术都需要多次迭代来实现,时间复杂度过高,而且结果不保留顶点到分区的映射,因此并不适用于BSP模型下的数据划分。

另外,在基于MapReduce[20]的并行处理中也存在类似的数据划分和负载均衡问题。Gufler等人[21]给出了解决MapReduce作业在处理偏斜数据时负载不均衡问题的方法。传统的MapReduce作业是n个分区对应n个Reducer,而文献[21]摒弃了这种传统的方法,为了达到负载均衡的目的,创建了比Reducer更多的分区,在Reduce阶段使用贪婪算法决定分区到Reducer的对应关系,将多个分区对应一个Reduce任务,使每个Reducer尽量负载均衡。这种算法每次只是将负载最大的分区分配给负载最轻的Reducer,并不能有效地保证每个Reducer完全均衡。

上面介绍的是与本文工作相关的图数据划分算法。它们要么不能实现满足BSP模型的图数据划分,要么不能实现数据的均衡划分。下面将介绍经典的Hash数据划分算法。

3　经典Hash划分算法介绍

经典Hash划分算法处理流程分为两步:

(1)从分布式存储系统读取数据。

(2)根据Hash函数计算一个顶点属于哪个分区。这里的Hash函数可以有多种选择,例如Java提供的Hashcode()函数,或者是MD5算法等。

采用Hash方式进行图划分的好处是划分算法简单,易于实现,同时很容易确定每个顶点属于哪个分区,而不需要维护一个巨大的路由表。但是,它也存在以下两点不足:

(1)对于海量的图数据往往会造成数据偏斜,这样会使一个任务分配了过多的数据,从而使这个任务的负载过重,产生“水桶效应”,使整个作业的运行时间由这一个任务的执行时间决定。如何避免负载不均衡是这个问题的根本所在。

(2)在使用经典Hash实现划分时,由于没有考虑数据的拓扑结构,数据可能被完全散列分布到所有节点上,完全打破了图的内在结构,很大可能将任务初始分配的分片中大部分数据散列到其他节点上,这就需要将大部分数据发送到其他参与运算的节点上,导致了因数据迁移而带来的通信开销。因此要解决的问题是,在采用Hash方式划分时,如何将本分片上的数据尽可能地保留在本节点上,从而减小在数据加载时的数据迁移开销。

4　作业运行的代价模型

为了将图部署到分布式系统中,需要将大图数据划分为若干个子图,然后将每一个子图分配给集群中的某一个节点进行处理。图计算的基础是图遍历,而图遍历的基本特征是通过边对顶点进行访问与遍历[2]。因此,跨越集群节点的边数决定了图处理时的网络通信开销。通常将存储在不同节点上的两个顶点连成的边称做交互边。访问非本地数据的网络开销非常大:访问本地内存数据的时间通常以纳秒计算,而访问非本地数据的网络通信时间则通常以毫秒计算,两者相差4~5个数量级[2]。因此,图划分的方式直接决定了交互边的数量,从而决定了基于该划分方式的图计算所需的通信代价。因此,在保证各个分区中图顶点数目尽量均衡的前提下,希望通过减少子图间的交互边数量来降低通信代价。

这里将作业定义为用户提交的需要解决的问题,它由处理程序、数据、处理说明及相关配置组成。一个作业由多个任务完成,每个任务完成对一个作业的一部分数据的处理。一个作业的时间代价通常由四部分组成:作业初始化加载数据的时间,本地计算用时最长的那个任务所花费的时间,任务之间进行全局通信的时间开销,在超步最后进行路障同步所花费的时间。其中,第一部分时间开销只在作业开始时存在,而后三部分时间开销在每个超步中都存在。

数据划分后,在数据加载过程中需要进行数据的迁移,将非本节点上的数据迁移到本节点上。用R_i表示节点i需要迁移的记录条数,S_j表示j所含顶点数,一个顶点所占的字节数为B,用$Time$表示迁移每字节所用的时间,则对于每个节点迁移数据的时间代价$Cost_1(i)$为:

$$Cost_1(i)=Time\times\sum_{i=0}\sum_{j=1}^{R_i}S_j\times B \tag{1}$$

由于各任务并行执行,因此加载数据的时间由运行时间最长的任务决定。对于一个有 n 个任务的作业来说,加载数据所用时间为:

$$W_{\mathrm{L}}=\max_{i=1}^{n}(Cost_1(i)) \tag{2}$$

这里,T 表示向非本地节点发送每个数据包所需的网络通信时间;t 表示访问本地内存数据每字节所需时间。对于每个分区 i,出边顶点在非本地的有 $Eout(i)$ 个,出边顶点在本地的有 $Ein(i)$ 个,顶点在网络间传输以数据包为单位,每个数据包中包含 1 000 条数据,在本地的顶点则直接将消息发送给相应的顶点而不需要网络通信。设一个顶点所占的字节数为 b,则对于每个节点一个超步中访问数据的时间代价 $Cost_2(i)$ 为:

$$Cost_2(i)=T\times\left\lceil\frac{Eout(i)}{1\,000}\right\rceil+b\times t\times Ein(i),T>>t \tag{3}$$

由于在一次迭代中各任务是并行执行的,因此一个超步下通信开销由花费最大的那个任务决定。对于一个有 n 个任务的作业来说,第 k 个超步的网络通信时间开销为:

$$W_{\mathrm{IO}}^{(k)}=\max_{i=1}^{n}(Cost_2(i)) \tag{4}$$

一个超步的本地计算时间开销由耗时最长的那个任务决定,设每个任务本地计算时间开销为 w_i。那么,对于一个有 n 个任务的BSP作业来说,第 k 个超步中本地计算花费的时间为:

$$W_{\mathrm{C}}^{(k)}=\max_{i=1}^{n}(w_i) \tag{5}$$

由此可知,一个运行了 s 个超步的BSP作业的时间开销为:

$$W=\sum_{k=1}^{s}W_{\mathrm{C}}^{(k)}+W_{\mathrm{L}}+\sum_{k=1}^{s}W_{\mathrm{IO}}^{(k)}+l\times s \tag{6}$$

其中,l 为在每个超步最后进行路障同步所花费的时间。

从代价公式(1)可以看出,非本节点的数据越少,迁移数据的时间代价 $Cost_1(i)$ 越小。由此,在设计划分算法时要考虑数据的本地化,将本分片的数据尽量保留在本节点上,从而减少数据的迁移量,降低节点迁移数据的时间开销。

从代价公式(3)可以看出,如果从出度边的角度进行图的划分,最小化分区间的交互边,可以有效地减少网络通信的时间。由于 T 远大于 t,当 $Eout(i)$ 减小时,可以保证时间代价 $Cost_2(i)$ 明显减小,从而使 $W_{\mathrm{IO}}^{(k)}$ 减小。因此,在设计划分算法时要考虑分区间的交互边,最小化交互边的数量,从而减少计算过程中的通信时间。

基于对上述代价模型的分析,本文提出了实现负载均衡的Hash数据划分算法。

5 负载均衡的Hash图数据划分算法

5.1 BHP算法简介

基于对代价模型的分析,本文提出了负载均衡的Hash图数据划分算法。针对Hash算法的数据偏斜问题引入了虚拟桶的概念:将 m 个实际分区中每一个都扩大为 n 个分区。这样,整个系统具有 $n\times m$ 个分区,把这些分区称为虚拟桶,并存在一张映射表,用于记录一个实际分区对应着哪几个虚拟桶。图1和图2对比了传统数据划分算法与BHP算法的实现思想。

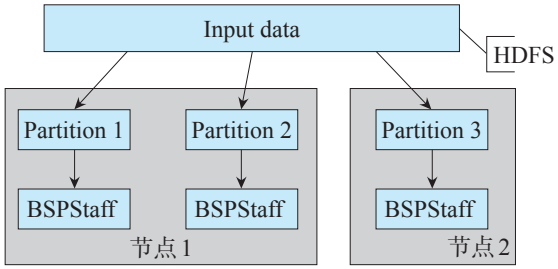


Fig.1 Partition idea of traditional algorithm

图1 传统算法分区的实现思想

假设一个集群有2个节点,每个节点最多可以处理2个BSP任务,一个任务对应一个分区。图1展示了传统算法的分区思想:有3个任务就把输入数据划分为3个分区,分区与任务一一对应。图2展示了BHP算法的基本思想:要将一个作业划分为3个分区,称之为实际分区。在得到实际分区前,先将数据划分成比实际分区多的分区,例如6个,这里每个分区称为一个虚拟桶。然后,再根据一定的规则将虚拟桶进行组合,形成实际分区。如图2中将6个虚拟桶组合成3个实际分区。因此,BHP算法的处理过程如下:首先,根据实际分区数设定虚拟桶数,利用Hash方法将数据划分到虚拟桶中,并统计出各虚拟桶内的数据量(这里主要是图的总边数,及其出边表)。然

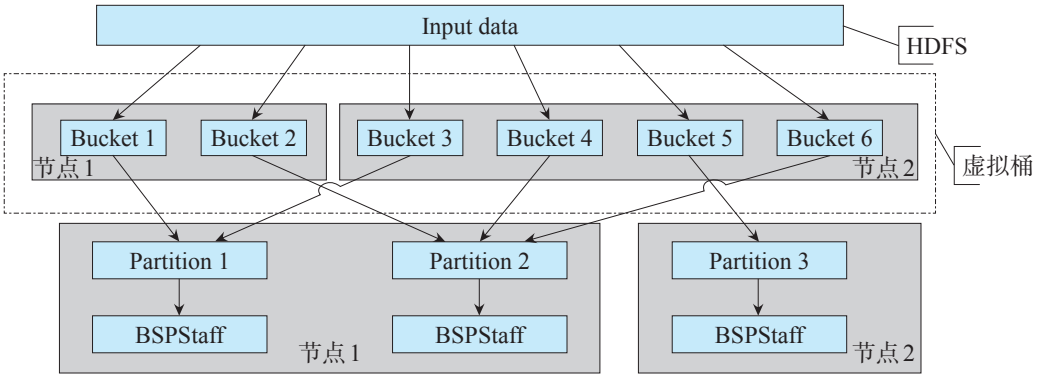


Fig.2　Partition idea of BHP algorithm

图2　BHP算法分区的实现思想

后,按照5.2节的算法和规则将虚拟桶组合成实际分区。重组时以虚拟桶的总出度值为参考,尽量将交互边较多的几个虚拟桶划分到一个实际分区中,并且保证每个由虚拟桶重组而得到的实际分区中数据量近似等于总数据量的均值。最后,根据第一步的Hash函数和虚拟桶的组合关系,将实际数据分配给一个实际分区,并分发给一个节点,从而实现数据的均衡划分。重组时充分考虑了虚拟桶之间的连通性,减少了实际分区间的交互边,从而减少了计算过程中消息的发送量,加快作业的完成速度。

上述数据划分过程尽量实现了各个分区的数据均衡。在此基础上,组合虚拟桶时考虑数据迁移问题。假设一个实际分区由 w 个虚拟桶组合而成,它们有的来自本地节点的虚拟桶,有的来自其他节点上的虚拟桶。如果能够统计出 w 个虚拟桶中的数据来自哪个节点的最多,就将这 w 个虚拟桶映射到那个节点上,从而减少数据加载时的数据迁移量。

5.2　BHP算法详述

基于上述讨论,BHP算法主要分为三个步骤:确定虚拟桶数目和虚拟桶划分,重组 $n\times k$ 个虚拟桶为 k 个实际分区,数据本地化。

(1)确定虚拟桶数目和虚拟桶划分

根据本文的计算框架,k 个分区会启动 k 个任务进行并行的分布式处理。因此图划分就是将图数据划分为 k 个分区,并且保证每个分区中分得的数据量基本一致。BHP算法引入虚拟桶的概念,虚拟桶的数目通常是实际分区数的 n 倍 $(n>1)$,n 值由实验获得。因此,根据设计的Hash函数将数据划分成 $n\times k$ 个虚拟桶。

(2)重组 $n\times k$ 个虚拟桶为 k 个实际分区

将数据划分到虚拟桶的过程只是实际划分的预处理,因此并没有实际地将数据分发到各个节点上,只是用设计的Hash函数计算出各个数据映射到哪个虚拟桶,然后将各个虚拟桶映射到某个实际分区,并统计各个虚拟桶中有多少数据。假设 P_i 表示预处理时虚拟桶 i 所在的实际分区号,虚拟桶 i 到各实际分区 P_j 的出度值为 E_{ij},每个虚拟桶的总边数为 E_k。当该虚拟桶到某个实际分区 P_j 的出度值大于该虚拟桶的总边数 E_k 的某个比率 α,即当 $E_{ij}\geqslant\alpha\times E_k$ 时,则将该虚拟桶 i 到实际分区的映射关系更改为 $i\rightarrow P_j$。同时将此虚拟桶标记为已放,并记录每个实际分区已放入的数据量,剩余未放入的虚拟桶采用基于贪婪算法的组合算法进行重组,每次将最接近实际分区剩余容量的虚拟桶装入实际分区中,不断循环直到所有虚拟桶全部放入为止。

假设一张图中一共有 e 条边,则每个实际分区分得的数据量平均情况应为:

$$avg(i)=\frac{e}{k} \tag{7}$$

最优的情况是每个实际分区 i 分配的数据量为 $avg(i)$,即均值。实际上要实现绝对的均衡划分是很难的,通常允许有一定的偏差。因此重组算法就是实现一种将 $n\times k$ 个虚拟桶组合为 k 个实际分区,且各个分区中的数据量尽量接近 $avg(i)$。因此,虚拟桶

重组算法如下所示。

算法1 BHP算法中的虚拟桶重组算法

功能:将 $n\times k$ 个虚拟桶重组为 k 个实际分区。

输入:counter,各任务的虚拟桶出边数量计数器;

maxBucketToPartition[buckets][2],各虚拟桶到各分区的数据量最大值及该分区ID。

输出:BucketToPartition,重组后的虚拟桶到实际分区的映射表。

1. 将counter中的*key*放到index数组,*value*放到value数组;
2. *avg*← 分区数据量均值;
3. for *i*←0 to 虚拟桶数
4. if 虚拟桶*i*到分区的最大出度>=*α*×虚拟桶*i*的总边数
5. if 分区容量−*value*[*i*]>0
6. 分区容量增加*value*[*i*];
7. 虚拟桶*i*标记为已放;
8. 添加虚拟桶到分区的映射表BucketToPartition;
9. 调用算法2确定剩余虚拟桶到分区的映射;
10. return BucketToPartition;

算法2 基于贪婪算法的虚拟桶重组算法

功能:基于贪婪算法将虚拟桶分配到 k 个实际分区中。

1. for *i*=0 to 分区数
2. while(!stop)
3. 从value数组中找到最接近 *avg*−*cap*[*i*] 的虚拟桶ID;
4. if(虚拟桶未放)
5. 分区容量增加*value*[*i*];
6. 虚拟桶标记为已放;
7. 添加虚拟桶到分区的映射表BucketToPartition;
8. else
9. stop=true;

(3)数据本地化策略

上述重组算法首先汇总了虚拟桶中数据的总量,根据这个总量将虚拟桶重组为实际分区。因为处理任务是独立分布式的,所以一个实际分区上的数据可能来自多个任务。但是只可以将一个实际分区分配给一个任务,根据代价公式(3),需要汇总各个任务上同一个实际分区的数据量,统计出该实际分区来自哪个任务的数据最多,则将该实际分区分配给那个任务。这样,得到优化后的映射关系表。本地化后减少了数据发送量,保证了原来在同一个分片上的数据还是属于该任务,同时图的内部结构得到一定的保留。实现数据本地化的算法如下所示。

算法3 BHP算法中的数据本地化算法

功能:统计组成实际分区的数据来自哪个任务最多,并把该实际分区分配给这个任务。

输入:partitionHasEdgeInStaff[][],每个实际分区来自每个任务的数据量。

输出:HashBucketToPartition,优化后的映射关系表。

1. for *i*←0 to 分区数
2. for *j*←0 to 任务数
3. if 任务*j*中分区*i*的数据量>*Max* && 任务*j*未分配
4. *Max*=partitionHasEdgeInStaff[*i*][*j*];
5. 更新分区ID为*j*;
6. 标记任务*j*为已分配;
7. 遍历每个虚拟桶
8. 更新虚拟桶到分区的映射;
9. return HashBucketToPartition;

负载均衡的Hash数据划分算法对于由MD5哈希、SRC32等经典哈希带来的数据偏斜状况具有良好的改善。但是如果指定的Hash算法本身就将数据完全映射到某几个分区,那么即使采用此算法进行优化也不可避免地会发生数据偏斜。因此它的平衡效果也取决于使用的Hash函数的设计。但是在使用相同Hash函数的情况下,此算法在实现效果上要比完全散列好,性能也要优于完全散列方法。

6 实验

6.1 实验环境和测试数据

实验使用的是项目组开发的云平台下的BC-BSP大规模图处理系统[22],类似于Hama等基于BSP模型的系统。本系统部署在由10台PC机构成的集群上。1台用于主控节点和Zookeeper服务器,其余9台作为计算节点。其中集群中节点配置如表1所示。本文测试使用的数据来源于斯坦福大学的3个真实网络数据集,大小依次递增[23]。3个数据集分别作为系统的输入进行PageRank计算。数据的具体情况如表2。由于现存的基于BSP模型的大图处理系统大多使用

的是Hash划分算法,本文选择使用经典Hash算法进行对比实验。

Table 1　Cluster configuration

表1　集群配置

名称	说明
处理器	4×Intel® Core™ i3-2100 @ 3.10 GHz
内存	8 GB
硬盘	500 GB
系统	Red Hat Enterprise Linux6.1
Hadoop	Hadoop-0.20.2
Zookeeper	Zookeeper-3.3.2

Table 2　Overview of graph datasets

表2　图数据集概述

数据名称	数据大小	顶点数	出边数
cit-Patents	203 MB	2 089 345	18 608 293
LiveJournal social network	705 MB	4 847 571	68 993 773
Wikipedia page-to-page link	1.29 GB	5 716 808	130 160 392

6.2　实际分区扩大倍数测试

虚拟桶数目的多少影响了数据划分的均衡性,进而影响基于BSP应用作业的执行速度。因此,如何确定虚拟桶的数目是一个需要解决的问题。图3给出了不同虚拟桶数目下,PageRank算法的执行耗时。

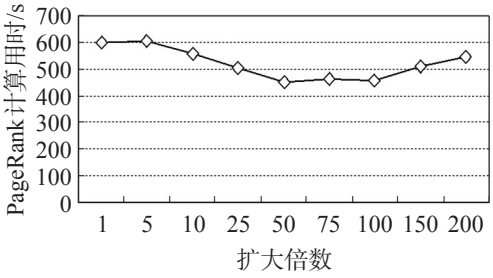


Fig.3　Relationship between n and the runtime of PageRank

图3　分区扩大倍数n与PageRank算法计算用时的关系

实验使用LiveJournal social network数据集,以PageRank算法为例进行测试。横轴表示实际分区扩大倍数,纵轴表示一次PageRank作业的运行时间。从图中可以看出当扩大倍数为50~100时计算用时最少。因此在以下实验中,取扩大倍数为50进行测试。

6.3　负载均衡测试

负载不均衡的程度可以刻画为划分不同分区规模的方差等统计指标。最大不均衡,定义为所有实际分区中的最大数据量超出最小数据量的差值。这里使用最大不均衡作为负载均衡测试统计量,分别统计了Hash算法与BHP算法的最大不均衡,结果如图4所示。

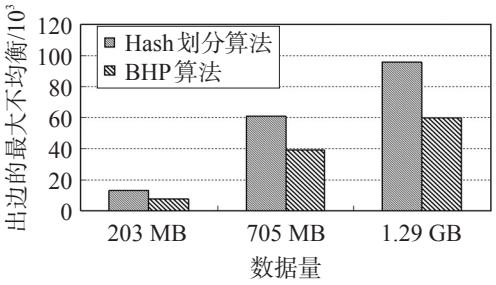


Fig.4　The largest imbalance of Hash and BHP algorithms

图4　Hash算法和BHP算法的最大不均衡

从图4中可以看出,BHP算法的最大不均衡相比于Hash算法明显减小,大约减少30%以上。这是因为BHP算法在重组时以虚拟桶的总出度值为参考标准,保证每个由虚拟桶重组后而得到的实际分区中边数相近,近似等于总边数的均值,所以分区间数据量相差不大。而Hash算法是完全散列的方法,并不均衡各个实际分区的数据量,因此会出现分区间数据量相差较大的情况。

6.4　通信代价测试

在进行数据划分时通信开销体现在两个阶段:其一是数据加载时采用某种方式进行划分会将本节点的一些顶点发送到其他节点上;其二是在图应用算法计算过程中需要进行消息通信。

图5给出了加载数据时两种算法的通信量,即节点间数据迁移量比较。可以看出Hash算法需要发送大量的顶点,而BHP算法由于采用重组的方法考虑了数据的本地化及图的拓扑关系,因而减少了对外发送的数据量。相对于Hash算法,BHP算法减少了数据迁移量,尤其是当数据量较大时可以减少30%以上的数据迁移量。

图6是两种划分算法在计算PageRank用例的一

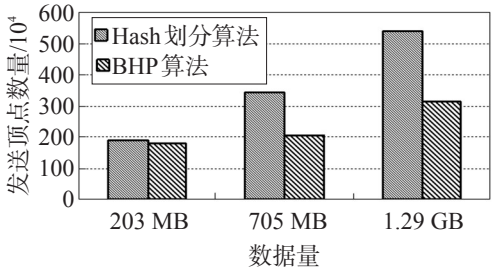


Fig.5 Sending vertex number of Hash and BHP algorithms at loading data stage

图5 Hash算法和BHP算法加载数据通信量

个超步中发送的消息数量。本文统计的是第二个超步。可以看出BHP算法相较于Hash算法发送的消息数减少25%以上。这是因为BHP算法在虚拟桶重组时,充分考虑了图的拓扑关系,减少了实际分区间的交互边,降低了消息的发送量。而经典的Hash算法将数据完全散列到所有节点上,打破了图的内在结构,因此增大了消息的发送量。

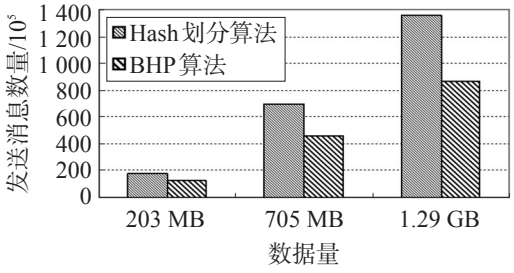


Fig.6 Sending message number per superstep of Hash and BHP algorithms

图6 Hash算法和BHP算法发送消息通信量

6.5 时间测试

时间测试主要测试两方面:一是划分算法加载数据用时;二是不同划分方法计算PageRank的用时。

图7是两种划分方法加载数据的用时对比。可以看出BHP算法加载数据时间大于Hash算法。这是因为BHP算法在加载数据阶段要进行优化重组处理和查找路由表的操作。

图8是采用上述两种划分方法完成一个PageRank计算作业的用时对比。可以看出BHP算法由于充分考虑了图的拓扑结构,各节点间交互边减少,保证了计算过程中发送的消息量降低,因此运行时间较少,尤其当数据量增加时每个超步用时降低很明显。

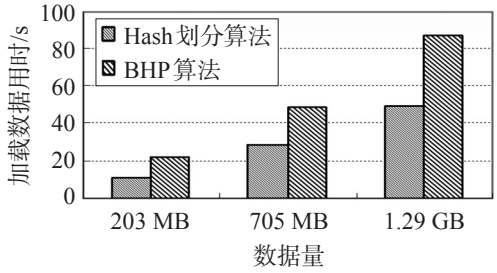


Fig.7 Time cost of Hash and BHP algorithms at loading data stage

图7 Hash算法和BHP算法加载数据用时

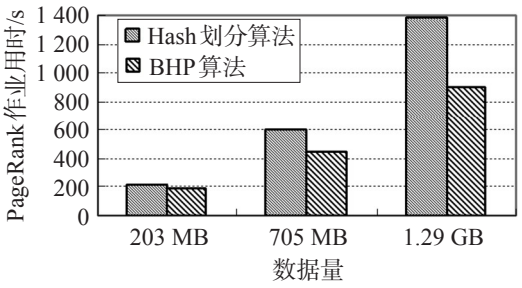


Fig.8 PageRank runtime of Hash and BHP algorithms

图8 Hash算法和BHP算法的PageRank运行时间

7 总结

由于图数据集的极度增长,使得在其上计算提取知识变得异常困难,传统的图处理工具已不再满足要求。近年来兴起的云计算技术已经为大数据集的处理提供了很好的支持。在集群环境下进行分布式并行大规模图数据处理中的一个重要问题就是图划分,而经典的图划分算法大都需要多次迭代,时间复杂度过高,而且划分结果不保留图顶点到分区的映射信息,因此并不适用于BSP模型下的数据划分。本文设计实现了面向BSP模型的负载均衡的Hash数据划分算法(BHP),此算法引入了虚拟桶的概念,在虚拟桶重组过程中同时考虑了分区的负载均衡和数据本地化,从而减少了消息发送的数量,提高了作业的执行效率。对比经典的Hash算法,BHP算法明显均衡了系统资源,提高了应用的执行速度。

References:

[1] Valiant L G. A bridging model for parallel computation[J]. Communication of the ACM, 1990, 33(8): 103-111.

[2] Feng Guodong, Xiao Yanghua. The distributed storage of

big scale graph[J]. Communications of the CCF, 2012, 8(11): 12-15.

[3] Kernighan-Lin algorithm[EB/OL]. (2011)[2013-02]. http://en.wikipedia.org/ wiki/KernighanLin_algorithm.

[4] Dutt S. New faster Kernighan-Lin-type graph partitioning algorithm[C]//Proceedings of the 1993 IEEE/ACM International Conference on Computer Aided Design (ICCAD '93), 1993: 370-377.

[5] METIS graph partition library[EB/OL]. [2013-02]. http://exoplanet.eu/ catalog.php.

[6] Mohar B. The Laplacian spectrum of graphs[M]//Graph Theory, Combinatorics, and Applications. [S.l.]: John Wiley, 1991: 871-898.

[7] Fildr M. Algebraic connectivity of graphs[J]. Czechoslovak Mathematical Journal, 1973, 23(98): 298-305.

[8] Page L, Brin S, Motwani R, et al. The PageRank citation ranking: bring order to the Web[EB/OL]. (1999) [2013-02]. http://www.diglib.stanford.edu/diglib/pub.

[9] Xiao Yanghua, Shao Bin. Managing and mining big graph data[R]. HotDB2012, 2012.

[10] Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large scale graph processing[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10). New York, NY, USA: ACM, 2010: 135-146.

[11] Apache Hama[EB/OL]. [2013-02]. http://incubator.apache.org/hama/.

[12] Apache Incubator Giraph[EB/OL]. [2013-02]. http://incubator.apache.org/giraph/.

[13] Low Y, Gonzalez J, Kyrola A, et al. GraphLab: a new framework for parallel machine learning[C]//Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI '10), 2010: 340-349.

[14] Low Y, Bickson D, Gonzalez J, et al. Distributed GraphLab: a framework for machine learning and data mining in the cloud[J]. Proceedings of the VLDB Endowment, 2012, 5(8): 716-727.

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

[16] White T. Hadoop: the definitive guide[M]. Zhou Minqi. Beijing: Tsinghua University Press, 2011: 600.

[17] Phothen A, Simon H D, Liou K-P. Partitioning sparse matrices with eigenvectors of graphs[J]. SIAM Journal on Matrix Analysis and Applications, 1990, 11(3): 430-452.

[18] Andersen R M, Peres Y. Local graph partitioning using evolving sets[J]. Foudations of Computer Science, 2006, 47: 475-486.

[19] Grivan M, Newman M E J. Community structure in social and biological networks[J]. Proceedings of the National Academy of Sciences, 2002, 99(12): 7821-7826.

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

[21] Gufler B, Augsten N, Reiser A, et al. Handling data skew in MapReduce[C]//Proceedings of the 2011 International Conference on Cloud Computing and Services Science (CLOSER '11), 2011: 574-583.

[22] Bao Yubin, Wang Zhigang, Gu Yu, et al. BC-BSP: a BSP-based parallel iterative processing system for big data on cloud architecture[C]//Proceedings of the Workshop on Big Data Management System, 2013.

[23] Mislove A, Marcon M, Gummadi K, et al. Measurement and analysis of online social networks[C]//Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC '07). New York, NY, USA: ACM, 2007: 29-42.

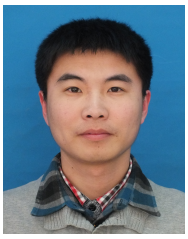
附中文参考文献:

[2] 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): 12-15.

ZHOU Shuang was born in 1988. She is a master candidate at College of Information Science and Engineering, Northeastern University. Her research interests include cloud computing and data mining, etc.

周爽(1988—),女,辽宁丹东人,东北大学信息科学与工程学院硕士研究生,主要研究领域为云计算,数据挖掘等。

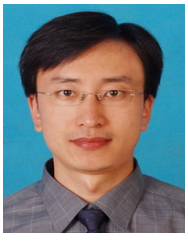
BAO Yubin was born in 1968. He received the Ph.D. degree in computer software and theory from Northeastern University in 2003. Now he is a professor at Northeastern University, and the senior member of CCF. His research interests include data warehouse, online analytical processing (OLAP), cloud computing and data intensive computing, etc.
鲍玉斌(1968—),男,吉林集安人,2003年于东北大学计算机软件与理论专业获得博士学位,现为东北大学教授,CCF高级会员,主要研究领域为数据仓库,联机分析处理,云计算,数据密集型计算等。发表学术论文30余篇,主持和承担过多项国家自然科学基金项目以及企业合作项目等。



WANG Zhigang was born in 1987. He is a master candidate at College of Information Science and Engineering, Northeastern University. His research interests include cloud computing and graph data mining, etc.
王志刚(1987—),男,山东烟台人,东北大学信息科学与工程学院硕士研究生,主要研究领域为云计算,图数据挖掘等。

LENG Fangling was born in 1978. She received the Ph.D. degree in computer software and theory from Northeastern University in 2008. Now she is a lecturer at Northeastern University, and the member of CCF. Her research interests include data warehouse and online analytical processing (OLAP), etc.
冷芳玲(1978—),女,辽宁锦州人,2008年于东北大学计算机软件与理论专业获得博士学位,现为东北大学讲师,CCF会员,主要研究领域为数据仓库,联机分析处理技术等。

YU Ge was born in 1962. He received the Ph.D. degree in computer science from Kyushu University of Japan in 1996. Now he is a professor and Ph.D. supervisor at Northeastern University, and the senior member of CCF. His research interests include database theory and techology, distributed system, parallel computing and cloud computing, etc.
于戈(1962—),男,1996年于日本九州大学计算机科学专业获得博士学位,现为东北大学教授、博士生导师,CCF高级会员,主要研究领域为数据库理论与技术,分布式系统,并行计算,云计算等。发表学术论文100余篇,主持和承担过国家自然科学基金、国家973计划、863计划等项目。



DENG Chao was born in 1978. He received the Ph.D. degree in computer science from Harbin Institute of Technology in 2009. Now he is a research professor at China Mobile Research Institute. His research interests include big data analysis and mining, distributed parallel computing, etc.
邓超(1978—),男,2009年于哈尔滨工业大学计算机科学专业获得博士学位,现为中国移动研究院云计算系统部研究员,主要研究领域为大数据分析与挖掘,分布式并行计算技术等。

GUO Leitao was born in 1983. He received the Ph.D. degree in computer science from University of Science and Technology of China. Now he is a research professor at China Mobile Research Institute. His research interests include NoSQL database, parallel computing and distributed storage, etc.
郭磊涛(1983—),男,中国科技大学博士,中国移动研究院云计算系统部研究员,主要研究领域为NoSQL,并行计算,分布式存储等。