基于图查询系统的图计算引擎

柯学翰，陈榕
上海交通大学软件学院并行与分布式系统研究所，上海 200240

摘要
在目前的研究中，图查询和图计算系统是相互独立的，但在实际应用中两者通常是同时存在的。为解决相互独立的系统带来的存储空间浪费、数据一致性维护等问题，基于图查询系统设计了一种图计算引擎，使得在单一系统中支持查询和计算操作。通过为键值对存储增加图计算索引、基于拉取模式的数据更新等方式，有效地提高系统中数据遍历的性能和减少数据传输的成本，同时针对数据更新和负载均衡等方面提出了相关优化。实验表明，该图计算引擎能够达到与传统图计算系统PowerLyra和Gemini相近或比其更优的性能，且具有较好的可扩展性。

关键词
分布式系统；图计算；图查询；键值对存储

中图分类号：TP311 文献标识码：A doi: 10.11959/j.issn.2096-0271.2019029

Graph processing engine based on graph query system

KE Xuehan, CHEN Rong
Institute of Parallel and Distributed Systems, Software School, Shanghai Jiao Tong University, Shanghai 200240, China

Abstract
Recently, graph query and graph processing are emphasis of graph-structured data research. However, independent graph system mismatched combining applications, which needed both query and processing. To avoid some issues brought by independent system, such as wasting resource and data inconsistency, a method that providing a graph processing engine based on graph query system was proposed, in order to support query and processing operation in a unified system. Through adding index for graph processing and applying pull-based graph propagation method to over locality issue, the performance of the computation and transmission was largely improved. Besides, some optimization approaches were put forward for message updating and work balanced. The experimental results show that the processing engine can provide close(reduced by no more than 1x) or even better (up to 20x) performance compared to specific graph processing systems (e.g., Gemini and PowerLyra) by leveraging new designs and optimizations, and also has good scalability.

Key words
distributed system, graph processing, graph query, key-value store

1 引言

近年来，随着互联网和社交网络的快速发展，大规模的图结构数据逐渐增多，例如将知识图谱、社交网络等信息抽象成的图结构数据。相比于传统的大数据处理系统，图系统能更好地利用图的结构信息，对图数据的处理更为高效。目前对图系统的研究可分为图查询系统和图计算系统两个方面。

图查询系统需要找到符合用户需求的图数据，常见的图查询系统有Wukong[1]、TriAD[2]、Trinity.RDF[3]等。图查询任务通常只需要访问全图中小部分的数据，但对时延非常敏感，需要在秒甚至毫秒级返回结果。因此，图查询系统通常使用键值对的存储模式，使得对单个顶点的访问更加高效。与图查询系统不同，在图计算系统中，一般使用稀疏矩阵存储图的结构。图计算任务通常需要访问全图上所有的顶点，对全图上的数据进行多轮迭代计算后才能结束，时延通常是分钟甚至小时级别的。因此，在图计算系统中，单个顶点的访问时延不是最重要的，其更关注的是整个系统的计算吞吐率。常见的图计算系统有Pregel[4]、PowerGraph[5]、PowerLyra[6]、Gemini[7]等。

目前对图查询系统和图计算系统的研究一般是相互独立的，但在实际应用中，图查询和图计算任务通常是同时存在的。例如对于一个记录了电商平台上用户和商品之间的关系的图数据，电商平台既有查询用户历史订单的需求（图查询任务），又有基于该图数据进行商品推荐的需求（图计算任务）。传统的做法是在图查询系统和图计算系统中分别加载该图数据进行分析。但是一份数据多份存储会带来许多的问题，例如内存空间的浪费、维护不同系统间数据的一致性等问题。

为了避免以上问题，本文在现有图查询系统基础上设计和实现了一种高效的图计算引擎，其能够在单个系统中同时支持高效的图查询和图计算操作。首先给键值对的存储结构增加针对图计算的索引，使其加快对图的遍历效率；其次针对图系统中的数据划分，为其设计了基于拉取（pull）模型的消息传递模式；最后针对该计算引擎的数据更新和负载均衡等方面进行了优化。在不同的测试集中的测试结果表明，该计算引擎图计算性能可达到PowerLyra系统的4.7倍到20倍，同时具有良好的可扩展性。

2 背景介绍

2.1 图数据的存储结构

键值对存储因具有可扩展强、结构简单、查找迅速等特点被广泛应用于图查询系统中，如Wukong[1]、Trinity.RDF[3]。在Wukong系统中，图上的边会转换成键值对进行存储，将顶点编号、边的类型、边的方向、值的地址和大小等信息组合成键（key），对应邻居顶点构成值（value），如**图1**所示。当需要查询顶点1、边类型为2的所有入边（in）时，先通过Hash函数找到对应的键的存储位置，然后根据键得到值的存储地址（offset），最后再通过远端或者本地访问的方式获取值的信息，即对应的邻居有顶点8和顶点9。

在图计算系统中广泛使用压缩稀疏矩阵来存储图的结构，如**图2**所示，包括GraphLab[8]、PowerGraph[5]、Gemini[7]等系统。行压缩稀疏矩阵（compressed sparse row，CSR）表示出边的信息，

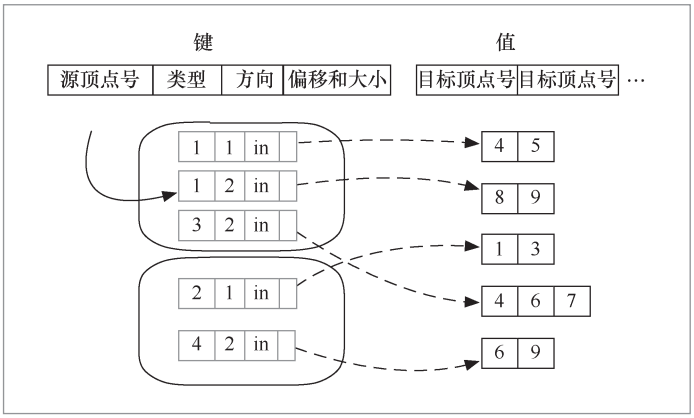

图 1 键值对存储

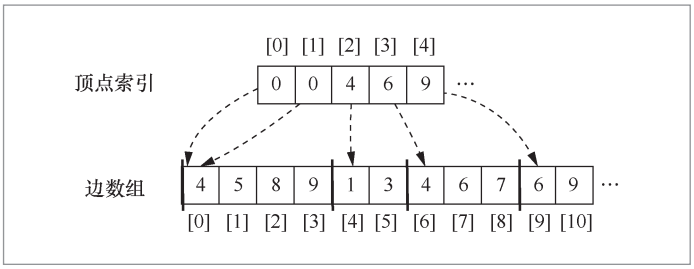

图 2 压缩稀疏矩阵存储边的数据

列压缩稀疏矩阵（compressed sparse column，CSC）表示入边的信息。顶点索引（vertex index）记录了每个顶点在边数组中的起始位置，并且顶点编号与顶点索引数组的序号保持一致。如顶点2，在顶点索引中的值为4，则顶点2的邻居顶点从边数组中下标为4的元素开始，一直到下一个顶点对应的索引值6，也就是说顶点1、顶点3是顶点2的邻居顶点。若该结构为CSC，则（1，2）和（3，2）是原图中的边；若为CSR，则（2，1）和（2，3）为原图中的边。压缩稀疏矩阵的图存储方式对于遍历图上所有边的计算而言是高效的。

2.2 图计算系统的图划分和执行模式

在图计算系统中，图划分在减少数据跨机器通信、负载均衡等方面发挥着很重要的作用。目前的划分方式可以分为边划分（edge-cut）和点划分（vertex-cut），如图3所示。

边划分[1,8-9]是指图从边切开，每个顶点被放置在一台服务器上（通常通过Hash的方式），也就是该顶点对应的边信息都存储在该机器上，其他服务器上只有该顶点的镜像顶点，因此每条边会在多台机器上出现。边划分的优点是计算过程中对邻居顶点信息的聚集都可以在本地完成；缺点是对于幂律分布的图，会出现负载不均衡的问题。幂律分布的图的特点是少部分的点拥有大量的边，因此拥有着这些点的机器的信息计算和通信开销会远大于其他的机器。点划分[5]是将每条边唯一放置在一台机器上，顶点可能会被切分在不同的机器中。点划分的优点是对于幂律分布的图也能实现很好的负载均衡。但是存在的问题是，在计算的过程中，由于一个顶点被切分在不同服务器上，则聚合邻居顶点的信息需要进行跨机器通信。还有一些工作[6]是将点划分和边划分的方法相互结合，为图上不同的顶点提供不同的划分方法。

图计算引擎的实现通常有两种方式：基于推送（push-based）模式[4,10-11]和基于拉取（pull-based）模式[12-14]。基于推送模式是对源顶点进行遍历，然后源顶点将自身的状态通过出边更新邻居顶点的状态。相反地，基于拉取模式是对目标顶点进行遍历，通过入边拉取邻居顶点的状态更新自己。相比于基于推送模式的更新邻居顶点（写）操作，基于拉取模式的引擎只需要拉取邻居顶点的信息（读）即可，因此其能够达到更高的计算吞吐率。基于推送模式比较适合图中活跃顶点较少的算法，可以方便地跳过该轮迭代中没有活跃的顶点，减少计算量。同时也有系统混合使用了两种更新方式，在执行的过程中动

态地选择适合的更新模式，如Gemini[7]、Polymer[14]等系统。

3 图计算引擎的设计和优化

该节主要介绍了如何在图查询系统中设计和实现一个高效的图计算引擎。首先总结了在图查询系统上实现图计算引擎的两点挑战；然后针对两点挑战分别提出了针对图计算索引优化和基于拉取模式的消息传递模式两种技术；接着介绍了图计算引擎的编程接口；最后给出了两种图计算引擎的优化方法：非阻塞式更新和负载均衡。

3.1 挑战

在单一系统中，所有的设计是为了该类型系统而设计的，包括数据的存储结构、数据的传输模型等。因此，不同系统间的设计是不匹配的，甚至是相互冲突的。首先，不同的系统对底层存储结构的要求不同。图查询系统一般使用键值对的方式存储图的结构信息，这样的存储方式有利于特定数据的快速查找，同时具有良好的可扩展性。而在图计算系统中，为了提升计算性能，需要的是支持高效图数据遍历的存储结构，例如CSR和CSC。其次，图计算系统进行数据传输的模式在很大程度上取决于图数据的划分方式。在一个图查询系统的数据划分方法下，一般不能直接套用现有图计算系统的数据传输模型，因为会出现顶点或者边的信息缺失等问题。

本文基于目前性能出色的分布式图查询系统Wukong[1]实现图计算引擎。键值对的存储结构具有很好的可扩展性，因此笔者希望在不改变原来图查询系统的基本的数据存储模式的情况下，增加高效的图计算引擎支持。基于以上分析，目前面临

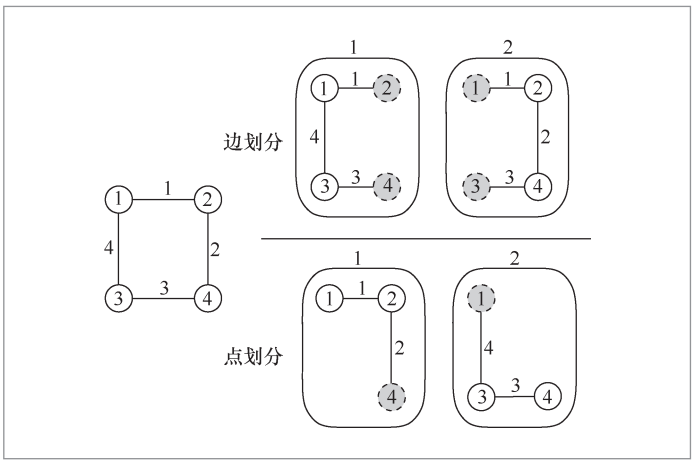

图 3 边划分和点划分

的挑战主要有以下两个方面。

挑战1：图计算系统需要高效的图遍历存储结构，如何针对键值对的存储进行高效的图计算。

直接使用键值对存储进行图计算存在的问题是计算性能不理想。在Wukong中，每个顶点访问其邻居顶点的信息时需要先构造对应的键，然后通过Hash表查找，最后才能获得邻居顶点的存储位置。这主要是因为图查询任务对于顶点的访问是随机的，Hash表可以加速一次随机的查找。而在图计算系统中，对于顶点的访问是顺序遍历的。CSC或CSR存储模式不仅可以通过一次访存操作获得邻居顶点的地址，而且使得数据具有很好的空间局部性。相比之下，使用Hash表查找的方式顺序遍历所有的顶点无疑是比较低效的。针对该问题本文提出了针对图计算的索引优化技术。

挑战2：图计算的数据传递模式在很大程度上取决于图数据的划分，如何在图查询系统中为图计算引擎设计合适的数据传递模式。

在Wukong系统中，非查询索引部分的图数据是按照边划分的模式进行的，即

每个顶点属于唯一一台机器，并且为了加速查询，边的信息会进行双向的存储。这种图划分的模式不同于PowerGraph[5]、Gemini[7]等图计算系统的划分方式，因此在Wukong系统上直接使用这些图计算系统的消息传递模式是不合适的。针对该挑战，本文提出了一种基于拉取模型的消息传递模式。

3.2 针对图计算的索引优化技术

为了解决第3.1节中的挑战1，图查询系统的存储结构需要支持高效的顺序遍历。高效的顺序遍历是指图系统能够快速地遍历图中所有的点和点对应的邻居，同时，原图查询系统的随机访问的性能不能受到影响。基于此目的，本文提出了针对图计算的索引优化技术。

针对图计算的索引优化是指在原先键值对的存储结构下，增加高效地顺序遍历索引的支持，使得顶点的遍历不需要通过Hash表获取顶点存储位置的地址偏移量，而是可以直接从索引中得到。这样能够大大地缩短数据访问的时间。

如**图4**所示，本文在原系统的存储结构中增加了索引的结构。原查询系统（Wukong[1]）中的数据存储结构主要包括

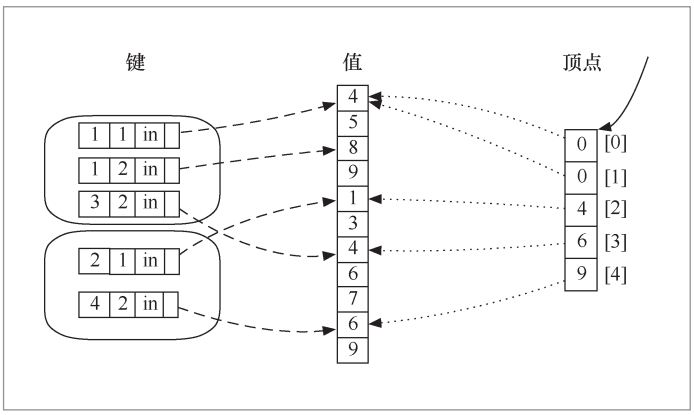

图4 基于图计算引擎的索引

两个部分：键存储和值存储。索引是一个数组的结构，数组的下标与对应的顶点ID一致，数组中的值为该顶点在值存储中的起始地址偏移量，对应的终止偏移量可以根据下一个顶点的起始地址偏移量来计算。例如，1号顶点对应的起始偏移量是0，2号顶点对应的起始偏移量为4，说明1号顶点对应的邻居顶点为值数组中0号到3号的位置的数，分别为4号、5号、8号、9号顶点。需要注意的是，在原图查询系统中，不同键对应的值的存储可以是不连续的。在新的存储模式下，为了便于索引的访问，值需要按顶点ID有序并且连续存储。但这样的限制不会对原先的图查询系统产生性能影响。

在增加了图计算索引的存储结构下，图数据的访问模式主要分为以下两种。

- 图查询任务：与原查询系统一致，首先通过Hash函数找到特定顶点的键的位置，然后根据键找到值的存储位置，即可获得邻居顶点的信息。
- 图计算任务：当图计算引擎需要遍历所有顶点的信息时，通过遍历图计算索引上的数据，就可以直接获得对应顶点的邻居信息的偏移量。

通过添加图计算的索引，图计算引擎对顶点的遍历基本与使用压缩稀疏存储结构一致，因此对图数据的访问也可以达到与单一图计算系统相似的性能。通过索引，对于每个顶点只需要一次内存访问就可以获得其对应的邻居顶点的偏移量。对于图的遍历，只需要顺序遍历一次索引数组和值数组即可，并且在计算过程中数据也具有很好的空间局部性。

3.3 基于拉取模式的消息传递

图计算引擎的消息传递模式与图的划分方式有很大的关系，因为图数据划分的

模式影响了顶点收集邻居顶点的消息来更新自己的方式。在Wukong中，键值对的存储模式事实上是一种边划分的方式，即每一个顶点只属于一台服务器，在其他服务器上的只是它的镜像顶点。

根据图查询系统的数据划分特点，本文使用基于拉取模式的消息传递，类似于Ligra[13]、Polymer[14]等系统中使用的pull模式。在每轮迭代中主要分为两个步骤进行，如**图5**所示。

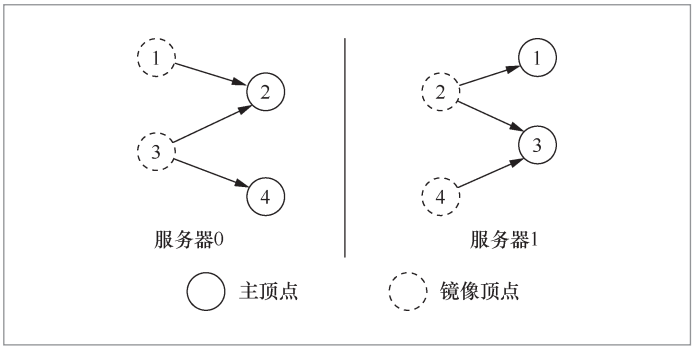

图 5　基于拉取模型的消息传递模式

步骤1　每台服务器上的顶点拉取其入边顶点的消息来更新自身的值。例如顶点2通过入边信息，聚合邻居顶点1、顶点3的值，然后更新自己的值。

步骤2　每台服务器上的顶点会将步骤1中更新的值发送给其他机器，更新其镜像顶点的值，到此一轮迭代的计算完成。例如服务器0上的顶点2、顶点4会发送信息给服务器1，以更新服务器1上顶点2、顶点4的镜像顶点的值。

在图查询系统Wukong中选择拉取模式而不是推送模式，是由其数据的存储模式决定的。因为每台服务器存储的信息是主顶点（master）聚集起来的，如果选择推送的模式，则每个顶点需要发送信息更新它的出边邻居顶点，发送的消息数量为$O(E)$（E表示边的数量，发送消息的数量与边的数量成正比）。例如服务器0上的顶点2需要更新服务器1上的顶点1、顶点3，因为服务器1上没有顶点2的邻居信息（只能通过顶点1、顶点3访问顶点2，不能通过顶点2访问顶点1、顶点3），因此服务器0需要发送两条信息，分别更新服务器1上的顶点1和顶点3。而在拉取模式下，顶点的聚合操作都是在本地进行的，不同服务器间只需要进行主顶点和镜像顶点的通信即可，消息发送数量由$O(E)$减少为$O(V)$（V表示顶点的数量）。

同时，对于不同机器间的顶点更新，本文采用了批量更新（batch）的方法，以减少单次数据更新的开销。批量更新是指将需要更新的顶点数据聚集在一起，然后一次性发送给其他的机器进行更新，而不是每个顶点单独发送一条更新消息。批量更新的方法虽然增加了单次数据发送的时间，但是大大地降低了数据发送的次数，因此平均下来每一条数据的传播时间被极大地缩短。

3.4 图计算模型抽象接口

本文借鉴了其他图计算工作[13-14]中提出的抽象接口，为用户提供了两种操作接口：Vertex_map和Edge_map。在接口设计上，保持了图计算系统中“像顶点一样去思考”的设计原则[15]，接口介绍如下。

- Vertex_map(F_Vertex, Active)：这个接口通过F_Vertex定义了单个顶点本地的操作。F_Vertex 为用户自定义函数，参数为当前顶点ID。用户可以自己定义如何对单个顶点进行操作。Active为活跃顶点的集合，每轮迭代中，只有活跃的顶点参与计算。
- Edge_map(F_Edge, Active)：这个接口通过F_Edge定义用户如何在边上进行数据聚集操作。参数F_Edge是一个用户自定义函数，该函数的参数为当前顶

点ID和该顶点所有入边顶点。

算法1给出了PageRank算法使用上述接口的具体实现。

算法1: Pagerank 算法。

```
D_next <- {0.0,0.0 … 0.0}
D_curr <- {0.0,0.0 … 0.0}
F_Vertex (v){
D_curr[v] = 1/|V|;
}
F_Edge(s,dst[]) {
 for(i =0;i <dst.size;i ++) {
D_next[s] <- D_curr[dst[i]]/Out[dst[i]];
        }
        D_next[s] <-0.15/|V| + 0.85*D_next[s];
}

PageRank（iter_num） {
iter<- 0
A<- V
        Vertex_map(F_Vertex,A);
   while iter<iter_num do {
            Edge_map(F_Edge,A);
            Swap(D_curr,D_next);
        }
}
```

3.5 优化

3.5.1 非阻塞式更新

在拉取模式下的步骤2，需要将本地更新的主顶点数据发送给其他机器，更新对应的镜像顶点。阻塞式更新是指服务器在接收别的服务器发送过来的更新数据时一直处于等待的状态，直到所有的数据接收完成后才开始本地的更新操作。

而非阻塞式更新在接受消息时不会阻塞整个更新的过程，即在接收数据的同时也在更新本地的数据。具体实现如图6所示，将数据接收和计算交由不同的线程负责。通信线程（communication thread）负责数据的接收，当接收一部分数据后就通知前台计算线程（computation thread）。计算线程发现有可更新的数据时，就将数据更新到本地，此时通信线程仍在继续接收新的数据，这样数据的接收和更新是并行的。当数据接收完成时，数据的更新也基本完成，使得消息传播的时间“覆盖”更新时间。

3.5.2 负载均衡

负载均衡是分布式并行计算系统一个重要的研究方向。对于一个同步的图计算引擎来说，计算的时间取决于最慢的机器的执行时间。其中，同步的图计算引擎是指新一轮迭代的开始需要等待所有的点完成上一轮迭代。因此，不同机器间以及单个机器中不同线程间的计算任务需要尽可能均衡。不同机器间的负载均衡由图的划分来保证，本文主要关注单台机器上不同线程间的负载均衡问题。针对该问题，笔者提出两个优化方案：基于边数量的任务划分和任务窃取。

基于边数量的任务划分方法是基于Grazelle[16]系统中的思想提出的，指依据边的数量为每个线程划分负责的点的数量。拉取引擎的计算过程包括两层循环，外层循环对所有目标顶点进行遍历，内层循环对每个目标顶点通过入边聚集源顶点的信息。不同的系统通常在外层循环中使用并行方法进行优化，即每个线程负责不同的目标顶点的计算。一种简单的划分策略是按照外层循环的顶点数量进行划分，但不同顶点对应边的数量不一致，这可能导致不同线程的计算量差异较大。因此，本文基于边数量预先为每个线程分配好需要负责的顶点。如图7所示，将下面的计算任务划分给两个线程，线程0负责0号顶点，

线程1负责1~5号顶点，每个线程中的计算都包含了7条边。如果使用基于点的数量的任务划分方法，则线程1负责0～2号顶点，一共10条边，而线程2负责3～5号顶点，一共4条边，会出现负载不均衡的问题。

任务窃取技术[17]被广泛应用在分布式并行系统中，它让已经完成任务的线程“窃取”其他线程未完成的任务来执行。在本系统中其可以与基于边数量的任务划分技术共同使用，具体实现如下：首先每个线程维护一个任务队列；然后将被分配好的任务划分成更多的子任务，保存在各自的任务队列里；最后每个线程从各自的任务队列里获取子任务并执行，当任务队列为空时，检查旁边线程的任务队列，“窃取”其他线程的任务来执行。

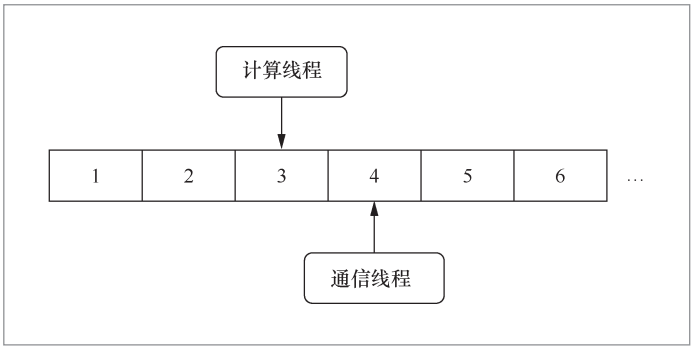

图 6 非阻塞式更新

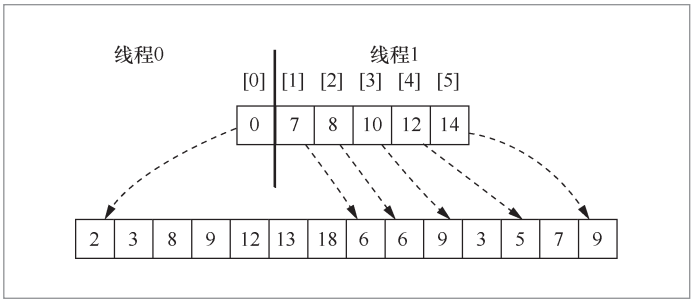

图 7 基于边的数量的任务划分

4 测试

PowerGraph[5]是一个功能完善、业界认可度比较高的图计算系统，PowerLyra[6]是在PowerGraph基础上针对幂律图进行改进的系统。Gemini[7]是目前性能比较出色的图计算系统，性能优于PowerGraph和PowerLyra。因此，本文选择PowerLyra和Gemini作为主要比较的系统。以下主要从性能和可扩展性两个方面对本文的图计算引擎进行分析。

4.1 实验环境

本文所有实验均在6台多核服务器组成的集群上完成，单节点配置如下：两个10核Intel(R) Xeon(R) E5-2650 v3 2.30 GHz处理器，内存分别为64 GB，其中远程直接内存访问（remote direct memory access，RDMA）网络使用ConnectX-3 MCX353A 56 Gbit/s InfiniBand网卡，通过Mellanox IS5025 40 Gbit/s交换机连接；以太网使用Intel X520 10GbE 网卡，通过Force10 S4810P 10GbE交换机连接。Wukong系统支持RDMA，因此在其基础上实现的图计算引擎使用RDMA进行机器通信，其他不支持RDMA的图计算系统使用普通以太网进行通信。**表1**给出了用于测试的数据集（UK、Twitter、RoadUS、Wiki）的相关信息，其中，$|V|$表示顶点数量，$|E|$表示边的数量。

表 1 实验数据集

数据集	$\|V\|$	$\|E\|$
UK	39×10^6	936×10^6
Twitter	41×10^6	1.46×10^6
RoadUS	23×10^6	58×10^6
Wiki	5×10^6	130×10^6

4.2 性能测试

图8是在4台服务器配置下，不同系统在多种数据集下的执行Pagerank算法（20次迭代）的时间对比。Pull-based表示在直接图查询系统中使用基于拉取模式的消息传递，没有使用其他的优化。Pull-optimal表示使用了相关技术优化后的图计算引擎，包括针对索引的优化技术、非阻塞式更新以及负载均衡等。其中Pull-based作为自身对照的基线系统，PowerLyra[6]和Gemini[7]作为与图计算系统对照的基线系统。

从**图8**可以看出，相比于自身基线系统Pull-based，Pull-optimal的运行速度是其2.08~3.14倍。在图计算任务中，对图数据的遍历访问主要集中在核心计算路径上，因此增加图计算的索引结构、加快图数据遍历速度可以极大地缩短图计算的整体执行时间。

相比于图查询系统PowerLyra，Pull-optimal的运行速度是其4.75~19.98倍。这一方面是因为PowerLyra是一个功能比较完善的图计算系统，其提供了更多的抽象和复杂图的操作，同时也带来了较大的开销；另一方面是因为在Pull-optimal中使用了高速网络RDMA，使得数据传输的时间大大缩短。相比于Gemini系统，在UK和RoadUS数据集下，本文中图计算引擎执行时间分别为Gemini的1.99倍和1.06倍。Gemini系统针对图存储结构做出了更多的优化，例如针对非统一内存访问架构（non-uniform memory access，NUMA）结构的存储、按块的图划分等。但是这些优化与现有的图查询系统存储是冲突的，不能应用于本文的系统上，因此Pull-optimal性能差于Gemini。在Twitter和Wiki数据集下，由于Gemini系统中机器间的通信数据量增大，占据了执行时间的绝大部分，而本文中图计算引擎使用高速网络RDMA，大大减少了网络的开销，因此性能优于Gemini系统。整体来看，本文基于图查询的图计算引擎相比独立的图计算系统，带来的额外开销不超过1倍，最优性能接近原性能的20倍。

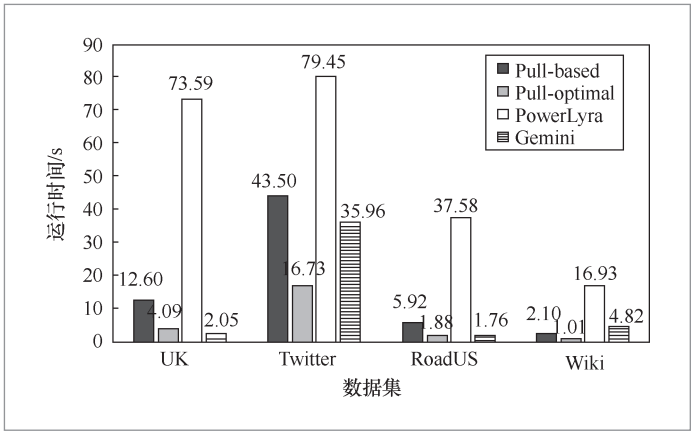

图 8 性能对比测试

4.3 可扩展性测试

图9展示了本文中图计算引擎随着服务器数目的增加整体运行时间的变化。测试使用Twitter作为测试的数据集，机器数量从1台变化到6台，运行PageRank算法。从**图9**中可以看出，该图计算引擎有很好的可扩展性。相对于2台机器（分布式模式下最小的机器数）的执行时间，当机器数目扩展到4台和6台时，分别可以达到2台机器性能的1.71倍和2.77倍。这是因为键值对的存储系统本身具有良好的可扩展性。

5 结束语

随着图结构化数据的增多，如何高效处理大量图结构数据成为研究的热点。但

由于目前相互独立的图查询系统和图计算系统与实际应用的需要不相符，本文提出了基于图查询系统的图计算引擎。首先通过为键值对存储添加图计算索引的方式，提高图计算的效率；其次，基于图系统中的图划分模式，使用基于拉取模式的消息传递；最后针对数据更新和负载均衡进行了优化。通过测试表明，本文提出的图计算引擎能够在兼容图查询系统的同时，利用各种优化技术提供与PowerLyra和Gemini接近或比其更优的性能，并具有较好的可扩展性。

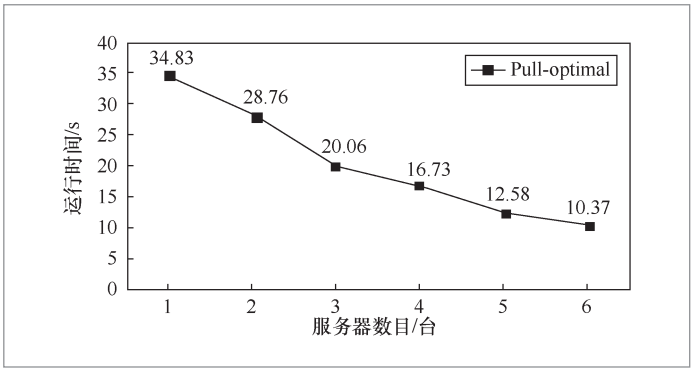

图 9 可扩展性测试

参考文献:

[1] SHI J, YAO Y, CHEN R, et al. Fast and concurrent RDF queries with RDMA-based distributed graph exploration[C]//The 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), November 2-4, 2016, Savannah, USA. Berkeley: USENIX Association, 2016: 317-332.

[2] GURAJADA S, SEUFERT S, MILIARAKI I, et al. TriAD: a distributed shared-nothing RDF engine based on asynchronous message passing[C]//The 2014 ACM SIGMOD International Conference on Management of Data, June 22-27, 2014, Snowbird, USA. New York: ACM Press, 2014: 289-300.

[3] ZENG K, YANG J, WANG H, et al. A distributed graph engine for web scale RDF data[J]. Proceedings of the VLDB Endowment, 2013, 6(4): 265-276.

[4] MALEWICZ G, AUSTERN M H, BIK A J C, et al. Pregel: a system for large-scale graph processing[C]//The 2010 ACM SIGMOD International Conference on Management of Data, August 11-13, 2009, Indianapolis, USA. New York: ACM Press, 2010: 135-146.

[5] GONZALEZ J E, LOW Y, GU H, et al. Powergraph: distributed graph-parallel computation on natural graphs[C]//The 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12), October 8-10, 2012, Hollywood, USA. Berkeley: USENIX Association, 2012: 17-30.

[6] CHEN R, SHI J, CHEN Y, et al. Powerlyra: differentiated graph computation and partitioning on skewed graphs[J]. ACM Transactions on Parallel Computing, 2019, 5(3): 13.

[7] ZHU X, CHEN W, ZHENG W, et al. Gemini: a computation-centric distributed graph processing system[C]//The 12th USENIX Symposium on Operating Systems Design and Implementation, November 2-4, 2016, Savannah, USA. Berkeley: USENIX Association, 2016: 301-316.

[8] 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.

[9] TSOURAKAKIS C, GKANTSIDIS C, RADUNOVIC B, et al. Fennel: streaming graph partitioning for massive scale graphs[C]//The 7th ACM International Conference on Web Search and Data Mining, February 24-28, 2014, New York, USA. New York: ACM Press,

2014: 333-342.
[10] WU M, YANG F, XUE J, et al. GraM: scaling graph computation to the trillions[C]//The 6th ACM Symposium on Cloud Computing, August 27-29, 2015, Kohala Coast, USA. New York: ACM Press, 2015: 408-421.
[11] ROY A, MIHAILOVIC I, ZWAENEPOEL W. X-Stream: edge-centric graph processing using streaming partitions[C]// The 24th ACM Symposium on Operating Systems Principles, November 3-6, 2013, Farmington, USA. New York: ACM Press, 2013: 472-488.
[12] WANG Y, DAVIDSON A, PAN Y, et al. Gunrock: a high-performance graph processing library on the GPU[J]. ACM SIGPLAN Notices, 2016, 51(8): 11.
[13] SHUN J, BLELLOCH G E. Ligra: a lightweight graph processing framework for shared memory[J]. ACM SIGPLAN Notices, 2013, 48(8): 135-146.
[14] ZHANG K, CHEN R, CHEN H. NUMA-aware graph-structured analytics[J]. ACM SIGPLAN Notices, 2015, 50(8): 183-193.
[15] MCCUNE R R, WENINGER T, MADEY G. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing[J]. ACM Computing Surveys, 2015, 48(2): 25.
[16] GROSSMAN S, LITZ H, KOZYRAKIS C. Making pull-based graph processing performant[J]. ACM SIGPLAN Notices, 2018, 53(1): 246-260.
[17] BLUMOFE R D, LEISERSON C E. Scheduling multithreaded computations by work stealing[J]. Journal of the ACM, 1999, 46(5): 720-748.

作者简介

柯学翰(1996-),男,上海交通大学软件学院并行与分布式系统研究所硕士生,主要研究方向为分布式图计算系统。

陈榕(1981-),男,博士,上海交通大学软件学院并行与分布式系统研究所副教授,主要研究方向为并行与分布式系统、内存计算等。

收稿日期: 2019-05-06
基金项目: 国家自然科学基金资助项目(No. 61772335)
Foundation Item: The National Natural Science Foundation of China(No. 61772335)