doi:10.11959/j.issn.1000-436x.2016066

基于 Graphlab 的网络图关键节点发现算法研究

高壮良[1]，吕雁飞[2]，张鸿[2]
（1. 北京航空航天大学计算机学院，北京 100191；2. 国家计算机网络应急技术处理协调中心，北京 100029）

摘 要：针对桥接中心度的计算特点设计了一种分布式的网络图关键节点发现算法（DABC），并基于 Graphlab 进行了实现。算法具有良好的扩展性，由于能够利用集群的内存资源，算法能处理的图规模与集群的大小成正比，并且该算法利用并行处理大幅度提升了计算速度。实验表明，与传统的基于单机实现的关键节点发现算法相比，算法可以获得高达 4 倍的性能提升。

关键词：关键节点；桥接中心度；分布式算法；Graphlab
中图分类号：TP316.4 **文献标识码：**A

Key nodes discovery in network graph based on Graphlab

GAO Zhuang-liang[1], LYU Yan-fei[2], ZHANG Hong[2]
(1. School of Computer Science and Engineering, Beihang Un versity, Beijing 100191, China;
2. National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing 100029, China)

Abstract: A distributed key nodes discovery algorithm was proposed(DABC) which was implemented on Graphlab. Due to the good scalability, the scale of graph supported by algorithm was enlarged significantly. The parallel processing also enhances the speed of calculation. Experiment results show that proposed algorithm can achieve up to 4 times performance improvement compared with the traditional centralized key node discovery algorithm.

Key words: key node, betweenness centrality, distributed algorithm, Graphlab

1 引言

网络图中的关键节点在图的组织和信息传播中起着枢纽作用，对图中关键节点进行标识和发现是图中的一个重要研究方向，有着丰富的应用场景和重要的应用价值。例如，社交网络成员中的关键节点通常具有更大的影响力或者更强的信息传播能力，找到社交网络中的关键节点可以分析甚至影响社交网络中的消息传播。在计算机网络拓扑中对关键节点进行保护可以提升整个网络的顽健性，反之对关键节点进行攻击会起到事半功倍的效果。此外，在反恐斗争中，关键节点研究也对重要恐怖分子的发现等[1]有着辅助作用。

识别网络图中关键节点的一类重要方法是计算图中节点的中心度，包括距离中心度、谱中心度和桥接中心度等。其中，桥接中心度是研究关键节点中常用的一种中心度度量，它可以用来衡量节点在网络图连通和信息传播中的重要程度。具有较高桥接中心度的节点代表着该点对图中的其他顶点的控制能力越大。近年来，桥接中心度在网络拓扑结构重要节点度量、关系网络研究等领域得到了广泛的应用。本文选用桥接中心度算法作为研究对象，下文如无特别说明，提到的中心度均为桥接中心度。

针对不同场景中的桥接中心度计算，相关工作已经给出了多种计算方法[2~9]。但是传统的计算桥接中心度的方法主要为集中式算法，即假设图存储在一台物理机上，并基于单机实现图的存储和计

收稿日期：2015-04-03；**修回日期：**2015-11-11
通信作者：吕雁飞，lyf@cert.org.cn
基金项目：国家重点基础研究发展计划（“973”计划）基金资助项目（No.2011CB302605）
Foundation Item: The National Basic Research Program of China (973 Program) (No.2011CB302605)

算。由于中心度计算算法的时间空间复杂度较高，集中式算法能够处理的图规模受到单机内存大小的限制，而且单机有限的处理能力也很大程度上影响了算法的执行效率。随着图规模的不断增加，算法的处理效率下降明显，严重影响了算法的应用范围，急需研究集群环境下的分布式算法。

分布式处理技术是目前的热点研究方向，以 MapReduce[10]为代表的分布式计算框架目前已变得非常流行，其在处理大规模数据方面有着相当多的应用。与此同时，一些分布式图计算框架也不断被提出，如 Pregel[11]、Graphlab[12,13]、PowerGraph[14]，GraphX[15]等。这些分布式计算框架为开发者提供了很多的 API 来设计和实现自己的算法。本文针对关键节点发现的分布式算法进行了研究。

本文设计了计算桥接中心度的分布式算法(DABC, distributed algorithm for betweenness centrality)。在 DABC 算法中为图顶点分布存储在多台物理机器上，DABC 算法设计了算法所需的数据结构，节点间数据通信机制，结果收集机制等。算法基于分布式图计算框架 Graphlab 进行了实现。由于可以使用分布式环境中多台服务器的内存资源，所以算法能够处理的图规模获得了较大提升，并且可以随集群规模扩展。同时由于使用了并行处理技术，算法的计算效率也比集中式算法明显增强。实验表明，在 8 台服务器组建的集群中，分布式算法可以处理 5 倍于集中式算法所能处理的图规模。由于支持并行处理，即使同在单机环境中，算法的处理速度也获得了提升，最好情况下提升幅度达到 4 倍以上。

2　相关工作

有向图中的桥接中心度是由 Anthonisse 等[2]在 1971 年首先提出。1977 年，Freeman[3]将这一概念引入具有不连通节点的网络中。现在使用最广泛的是 Freeman 提出的定义，即基于最短路径的桥接中心度算法。在这种定义中首先计算所有节点对之间的最短路径，在所有最短路径中次数较多的节点拥有较高的桥接中心度。Carpenter 等[4]在分析恐怖分子网络时指出桥接中心度是需要解决的重要的问题。Brands[5]通过对桥接中心度算法的深入研究，在 2001 年提出了高效的桥接中心度算法。在 Brands 提出的算法基础上，又产生了针对不同条件、不同类型图结构的变形算法[6]，包括 proximal betweenness 算法、bounded-distance betweenness 算法、distance-scaled betweenness 算法、group betweenness 算法等。2012 年，Lee[7]第一次提出了针对无向图的桥接中心度快速更新算法。同年，杨建祥[8]提出了社交网络桥接中心度快速更新算法。Tan[9]等提出了一种新的适用于 CREW PRAM 的并行桥接中心度算法，应用于大规模网络分析。这些算法的出现为分析图结构数据提供了方便，但是由于这些算法都未采用分布式处理技术，所以算法能够处理的图规模受到了较大限制，也影响了算法的应用场景。

近年来，利用分布式计算框架处理图数据已经越来越流行。分布式计算框架也在不断发展。Pregel[11]借鉴了 Leslie Valiant 于 20 世纪 80 年代提出的 BSP 计算模型，采用“计算—通信—同步”的模式完成机器学习的数据同步和算法迭代，计算由主机（master）负责分配任务和收集结果。同时采用了基于检查点（checkpoint）的系统容错方法。Giraph[16]是应用较为广泛的 Pregel 克隆版本，在 Pregel 基础上，增加了主节点计算、面向边的输入和核外计算等功能。Graphlab[12, 13]是最近比较流行的图处理框架，它是一个分布式异步内存共享系统，节点程序可以直接访问该节点、相邻边和相邻节点的信息，并通过阻止相邻的程序同时运行来保证结果的正确性。在 Pregel 和 GraphLab 的基础上，PowerGraph[14]特别针对社交网络图数据的幂律分布特性，借鉴了 GraphLab 的内存共享技术和 Pregel 的协同收集技术，提出了 Gather-Apply-Scatter 计算模型，分解高度数的节点并提供更大的并行性。

3　分布式关键节点发现算法

3.1　桥接中心度概念与计算方法

桥接中心度可以简单描述为网络图中的顶点在连接其他任意节点的最短路径中所占的比重。假设网络图表示为 $G=(V,E)$，其中，节点 V 用来表示图中的顶点集合，E 表示节点之间的边集合。在本文中只考虑无权图，即认为每条边的权值均为 1。对节点 $\forall s,t\in V, d_G(s,t)$ 表示 s 点和 t 点之间的最短路径的长度，若 $s=t$，$d_G(s,t)=0$。

对图 $G=(V,E)$，$\forall s,t\in V$，且 $s\neq v\neq t$，节点 v 的桥接中心度定义为

$$C_B(v)=\sum_{\forall s,t\in V}\frac{s_{st}(v)}{s_{st}},v\in V \tag{1}$$

其中，s_{st} 表示在有向图 G 中 s 点到 t 点的最短路径数量，$s_{st}(v)$ 表示在 s 点到 t 点的最短路径通过点 v 的数量。当 $v\in\{s,t\}$ 时，$s_{st}(v)=0$，若 s 点到 t 点不连通，则对 $\forall v\in V$，$\frac{s_{st}(v)}{s_{st}}=0$。

以图 1 中的有向图为例，图中包含 5 个顶点及 5 条边，顶点 b 处在 a 到 c，a 到 d，a 到 e 的最短路径上，且 a 到 c 及 a 到 d 的最短路径只有一条，a 到 e 的最短路径有 2 条，并且顶点 b 在 2 条最短路径上都有出现，所以顶点 b 的桥接中心度根据定义可计算得到。

$C_B(b)=1+1+\frac{2}{2}=3$，同理可计算其他顶点的桥接中心度大小为 $C_B(a)=C_B(e)=0$，$C_B(c)=C_B(d)=\frac{1}{2}+\frac{1}{2}=1$。将节点按照中心度大小排序，中心度越高的节点在网络图中关键程度也越高。图 1 中节点关键程度的排序为 $b>c=d>a=e$，这与直观观察和理解相一致。

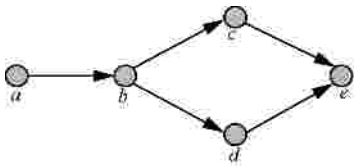

图 1 节点中心度示意

由节点中心度的定义可知，节点中心度计算可由计算图中两两节点的最短路径得到。目前计算中心度的最好算法是由 Brandes[5]在 2001 年提出的，本文中称为 FABC 算法(faster algorithm for betweenness centrality)。FABC 的计算过程可以简单描述如下。

定义点对之间的依赖为

$$d_{st}(v)=\frac{s_{st}(v)}{s_{st}} \tag{2}$$

将同一出发点的点对间依赖加起来，得到单点依赖 $d_s(v)=\sum_{t\in V}d_{st}(v)$，这样节点 v 的桥接中心度就可表示为

$$C_B(v)=\sum_{\forall s,t\in V}d_{st}(v),v\in V \tag{3}$$

而单点依赖满足

$$d_s(v)=\sum_{w:v\in P_s(w)}\frac{s_{sv}}{s_{sw}}(1+d_s(w)) \tag{4}$$

其中，$P_s(w)=\{u\in V:\{u,w\}\in E,d_G(s,w)=d_G(s,u)+c(u,w)\}$，表示顶点 s 到顶点 w 所有最短路径中顶点 w 的前继顶点集合。

对于无权图来说，该算法的时间复杂度达到 $O(mn)$，空间复杂度为 $O(n+m)$，其中，n 代表顶点数，m 代表边数。由于 m 的数量通常与 n 的数量成正相关，所以随着图中节点规模的增加，算法的时间与空间复杂度都会有显著增加。

在本文的实验环境中，对一个包含 3 000 个顶点的无向图，算法的计算时间约为 500 s，而且随着顶点数的不断增加，计算时间也随之增加。当顶点数达到万级别的时候，算法的时间消耗已变的不可接受。针对这个问题，本文提出了分布式的节点中心度计算方法。

3.2 DABC 算法概述

在分布式系统中，一个完整的图通常被切分为数个部分，由不同的机器进行存储和处理。实际应用的图数据多数呈幂律分布[17]，所以现在使用最广泛的切分方式是按点切分[18]。Powergraph 已经证明相比于边切分，点切分能带来存储、通信和计算上的优势。在点切分方式中，一个点被复制成几份分别存储在几台机器上，而边不做复制。本文把其中的一个顶点称作 master，而它的复制点统一称为 mirror。master 知道它的所有 mirror 信息，但是 mirror 只知道它隶属于 master 的信息。如图 2 所示，一个数据图被切分到 2 个机器上，顶点 c 与顶点 d 被切分，其中顶点 c、d 为 master，它们的 mirror 分别为 c_1、d_1。

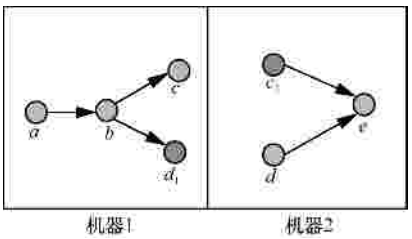

图 2 切分图

本文设计了分布式计算桥接中心度的算法。算法执行过程中，所有机器并行计算得到最终结果。以图 2 中的切分方式为例，图 3 介绍了算法的执行过程。

假设当前顶点 a 为执行顶点，顶点 b 接收到 a 的消息，将其到 a 的最短路径个数更新为 1，最短距离更新为 1（每条边权重默认 1）。如图 3(a)所示。

然后 b 将消息发给它的邻居 c 和 d_1，d_1 根据消息更新自己的 s 及 $dist$，d_1 将自己的数据发送给 d，顶点 c 根据消息更新自己的 s 及 $dist$，同时顶点 c 也将自己的数据同步到 c_1。直到顶点 e 收到来自 c_1 和 d 的消息，更新自己的 s 为 2 及 $dist$ 为 3，如图 3(b) 所示。

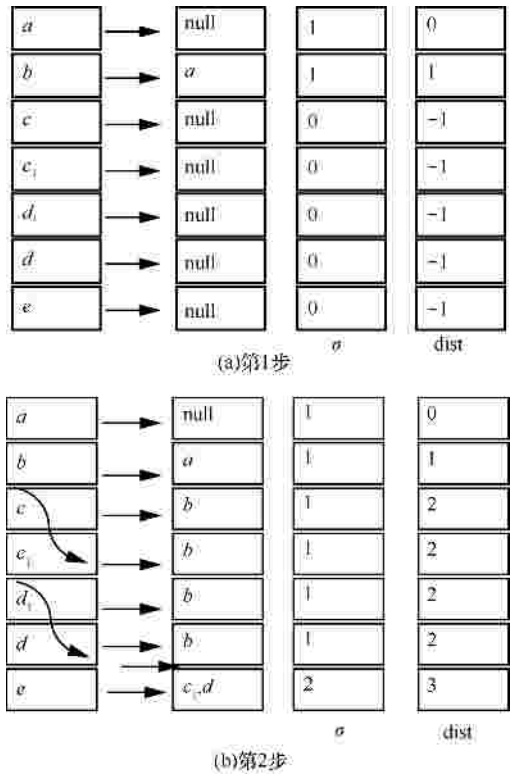

图 3　算法执行过程

最后根据式（1）可求得这一次消息传递中的各个顶点的桥接中心度。

3.3　基于 Graphlab 的 DABC 算法实现

下面将本节中用到的符号及其含义列出，如表 1 所示。

表 1　　符号及含义

符号	含义
V_p	在分区 P 中的所有顶点集合
V_P^{ma}	在分区 P 中的 master 顶点
V_P^{mi}	在分区 P 中的 mirror 顶点
$N_P^{out}(u)$	在分区 P 中顶点 u 的所有出边邻居
$N_P^{in}(u)$	在分区 P 中顶点 u 的所有入边邻居
$u_{dist}(v)$	顶点 v 到顶点 u 的距离，初始值- 1
$u_s(v)$	顶点 v 到顶点 u 的最短路径的数量，初始值 0
u_{done}	到顶点 u 的已计算过的顶点集合

本文使用了 Graphlab 框架来实现算法。Graphlab[12, 13]是基于 BSP 模型的图并行框架，以高性能著称。其不仅支持基于消息的编程模型，而且支持共享内存风格的“收集—更新—扩散”（记做 GAS）模型，除此之外 Graphlab 还支持异步计算，对自然图的并行处理具有较高的性能。因此本文以 Graphlab 为基础框架来实现。

Graphlab 主要分为 3 个计算过程。首先是收集阶段，工作顶点的边从邻接顶点和自身收集数据。然后是更新阶段，从节点将收集的数据发送给主节点进行汇总，主节点将汇总后的数据同步更新到从节点。最后是扩散阶段，工作顶点更新完成后，更新边上的数据，并通知对其有依赖的邻接顶点更新状态。

DABC 算法主要分为 2 个步骤。步骤 1：首先计算任意 2 个节点之间的最短路径及其数量，如图 4 所示。

/* V_P^{msg} 表示接收本地消息的顶点集合*/
1) for 所有的 $u \in V_p^{msg}$ do
2)　将本地消息求和保存在部分结果 u_{dist}, u_s 中
3)　if $u \in V_p^{mi}$ then 将 u_{dist}, u_s 发送到 master
4) 同步以保证所有机器均完成通信
5) for 所有的 $\frac{u \in V_P}{V_P^{mi}}$ do
6)　遍历 u_{dist} do
7)　if（$u_{dist}[w] < 0$） then
8)　$u_{dist}[w] = u_{dist}[u] + 1$
9)　if（$u_{dist}[w] = u_{dist}[u] + 1$）
10)　$u_s[w] = u_s[w] + u_s[u]$
11)　if u 有 mirror 顶点 then 将 u_{dist}, u_s 发送到所有 mirror 顶点
12) $\forall u \in V_P^{mi}$，if u 收到 u_{dist}, u_s ,then 更新本地信息
13) 同步以保证所有机器均完成通信
14) for 所有的 $u \in V_p$ do
15)　遍历 $w \in u_{done}$ do
16)　if $w \in v_{done}$ then
17)　发送 $\frac{u_{dist}}{u_{dist}[w]}, \frac{u_s}{u_s[w]}$ 至 $v \in N_p^{out}(u)$
18) 同步以保证所有机器均完成本轮迭代

图 4　算法的第一个分布式过程（一次迭代）

在步骤 1 的一次迭代中，主要包括 3 个超级步。

第 1 个超级步。1) 本地计算：工作节点 u 从接收到的来自本地邻居的消息（消息包含经过该邻居到节点 u 的当前最短路径及数量）中；2) 通信：如果 u 是 mirror，则将 u_{dist}、u_s 发送给它的 master；如果 u 是 master，则 u 接收其所有 mirror 的消息。

3) 同步屏障：确保所有 mirror 发送完消息并且 master 接收到发给它的消息。

第 2 个超级步。1) 本地计算每个 master 节点 u 从消息中遍历 u_{dist}，如果存在 $u_{\text{dist}}[w]<0$，那么令 $u_{\text{dist}}[w]=u_{\text{dist}}[u]+1$，如果 $u_{\text{dist}}[w]=u_{\text{dist}}[u]+1$，那么令 $u_s[w]=u_s[w]+u_s[u]$。2) 如果 master 发生更新，则将更新后的 u_{dist}、u_s 发送给其所有的 mirror；mirror 接收新的 u_{dist}、u_s，覆盖自己的 u_{dist}、u_s。3) 同步屏障，确保所有通信正常结束。

第 3 个超级步。对于发生更新的 master 和 mirror，将满足条件的 u_{dist}、u_s 发给各自的本地邻居，最后，使用同步屏障，确保所有操作结束。

步骤 1 完成之后，任意 2 个顶点之间的最短路径及其数量已经求得，然后利用公式便可求得每个顶点的桥接中心度，这是算法的步骤 2，如图 5 所示。

/* V_P^{msg} 表示接收本地消息的顶点集合*/

1) for 所有的 $u\in V_p^{msg}$ do

2) 将本地消息求和保存在部分结果 u_s 中

3) if $u\in V_p^{\text{mi}}$ then 将 u_d 发送到 master

4)同步以保证所有机器均完成通信

5) for 所有的 $u\in \frac{V_P}{V_P^{\text{mi}}}$ do

6) 遍历 u_d do

7) $u_d[v]=u_d[v]+\frac{u_s[v]}{u_s[w]}+(1+u_d[w])$

8) $C[u]=C[u]+u_d[u]$

9) if u 有 mirror 顶点 then 将 u_d 发送到所有 mirror 顶点

10) $\forall u\in V_P^{\text{mi}}$，if u 收到 u_d ,then 更新本地信息

11) 同步以保证所有机器均完成通信

12) for 所有的 $u\in V_p$ do

13) 遍历 $w\in u_{\text{done}}$ do

14) if $w=v$ then

15) 发送 $\frac{u_d}{u_d[w]}$ 至 $v\in N_p^{\text{in}}(u)$

16) 同步以保证所有机器均完成本轮迭代

图 5 算法的第 2 个分布式过程（一次迭代）

步骤 2 的一次迭代过程也包含 3 个超级步。

第 1 个超级步。1) 本地计算：工作节点 u 接收到来自本地邻居的消息（消息包含经过该邻居到节点 u 的点对依赖）。2) 通信：如果 u 是 mirror，则将 u_d 发送给它的 master；如果 u 是 master，则 u 接收其所有 mirror 的消息。3) 同步屏障：确保所有 mirror 发送完消息并且 master 接收到发给它的消息。

第 2 个超级步。1) 本地计算：每个 master 节点 u 从消息中遍历 u_d，然后根据式（3）及式（4）计算节点 u 的桥接中心度。2) 通信：如果 master 发生更新，则将更新后的 u_d 发送给其所有的 mirror；mirror 接收新的 u_d，覆盖自己的 u_d。3) 同步屏障：确保所有通信正常结束。

第 3 个超级步。对于发生更新的 master 和 mirror，将满足条件的 u_s 发给各自的本地邻居，最后，使用同步屏障，确保所有操作结束。

4 性能测试与分析

本节对分布式桥接中心度算法进行了测试和分析。测试的性能指标包括运行时间、内存消耗以及通信消耗。

4.1 实验环境

本实验共使用了 8 台物理服务器，服务器在内部局域网中通过吉比特交换机互连。服务器的硬件配置如下：CPU Intel Xeon E5-2650 2.0 GHz 8 Core，内存 8 GB，硬盘 500 GB。

实验使用的软件环境如下：操作系统 64 位 Debian7，Graphlab2.2，OpenMPI1.6，编译环境为 GCC 4.8，编程语言为 C++。

实验采用了真实数据集与模拟数据集 2 类数据进行评估。真实数据集来自 SNAP[19]，该数据集描述了维基百科中的查询请求关系。模拟数据集则以固定幂律生成了随机网络图数据。实验采用的数据集信息如表 2 所示。

表 2 实验数据集

数据集	顶点数	边数	边数 1	边数 2
数据集 1	1 000	—	4 186	10 006
	2 000	—	7 986	18 676
	3 000	—	21 231	25 679
	4 000	—	26 048	33 884
数据集 2	1 500	97 140	—	—
	SNAP10 835	159 388	—	—

数据集 1 中图的顶点数从 1 000~4 000，本文使用数据集 1 的边数 1 这组数据来测试本文提出的算法和原有算法在运行时间上的差别。同时本文也使用数据集 1 测试在固定顶点数量，变化边数量的情况下，算法的运行时间。

数据集 2 包含真实数据集 SNAP 和另一组模拟数据。数据集 2 比数据集 1 具有更大的数据规模，用来测试在多台机器上算法的运行指标。

实验比较了集中式的中心度计算算法和本文提出的分布式中心度计算方法 DABC 的性能，并测试了在多台服务器组成的集群环境下，DABC 算法的运行时间、内存消耗、通信量等指标。集中式的中心度算法采用了经典的 FABC 算法，在前文中已经有所介绍。

4.2　实验结果分析

图 6 给出了 FABC 算法与 DABC 算法的对比结果。测试使用了数据集 1 中的第 1 组数据。该实验在 1 台机器上进行，主要比较了算法在运行时间上的差别。如图 6 所示，本文提出的 DABC 算法比原有算法在时间上有了很大的提升，当顶点规模较小的时候，新算法的运行时间仅为原有算法的 25%以下，当顶点数达到 3 000 时，算法的加速比有所下降，但加速效果同样明显。进一步计算可知，在本实验中，新算法的平均加速达到 70%以上。

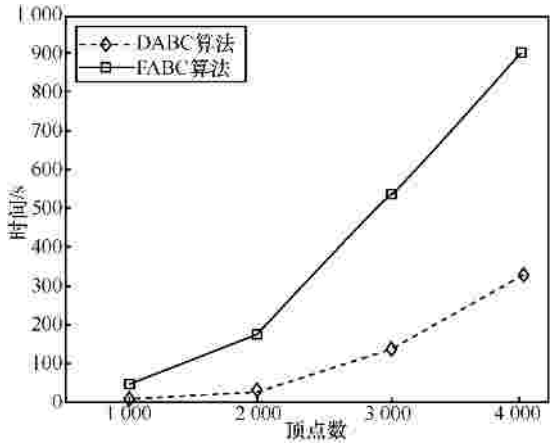

图 6　不同算法时间消耗对比

图 7 对比了在相同顶点数，边数不同的情况下 DABC 算法的运行时间。实验同样使用了数据集 1 中的数据。在数据集 1 中，本文固定图中顶点数，生成了不同边数的 2 组图。本实验中将边数较少的一组图称为稀疏图，边数较多的称为稠密图。

如图 7 所示，在相同顶点数的情况下，稠密图的耗时高于稀疏图。这是由于算法在运行过程中会搜索所有的边，所以边数的增加必然会导致运行时间的增长。

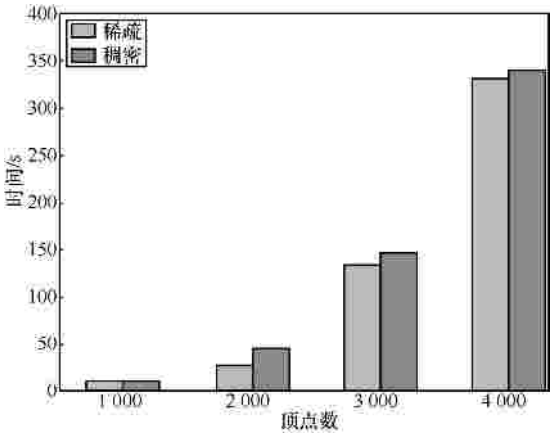

图 7　DABC 算法在稀疏与稠密图上时间消耗对比

在集群环境中，数据图会被切分到每台机器上存储。机器间通过通信协作完成统一的计算任务。切分方式在某种程度上决定了数据的分布方式和机器间的协作方式，所以一个好的切分方法将会使算法的性能得到很大提升。

在本实验中，对比了 3 种切分方式，分别为 random 切分方式、oblivious 切分方式、grid 切分方式。其中，random 切分方式将边随机的分配到每台机器上，切分速度较快，但是这种方式没有充分利用图的连通特征。而 oblivious 切分方式采取贪心算法策略进行数据图的切分，该切分方式解决的问题为在采用尽可能少的切分点的情况下达到负载均衡，该切分方式由于以贪心算法为基础，所以容易造成局部最优。grid 切分方式基于散列分配边，该方法切分速度快、切分均衡、并且切分点较少。

本实验使用了数据集 2 中的数据。如图 8 所示，本文分别在 2 台机器和 4 台机器上进行了测试。在 2 台机器的情况下，3 种切分方式的算法执行效率差不多，但是，随着机器数的增多，显然在 grid 切分方式下算法具有更好的效率。这是由于 grid 切分方式切分均衡、切分点少，使机器间通信减少，并且每台机器的计算负载也比较均衡。而另外 2 种切分方式，切分的点较多导致过多顶点被复制，使机器间通信频繁，而且切分不均衡也会导致集群的机器负载不匀，降低整体运行效率。

图 9 给出了算法在数据集 2 上的运行时间、内存消耗、通信量的统计情况。图 9（a）给出了算法在运行过程中的运行时间与机器数量的关系。随着机器数的增加，DABC 算法的运行时间不断降低，但是可以看到，加速率随着机器数的增加其有所下降。主要原因是由于机器数增加后，各机器间需要协作的工作也

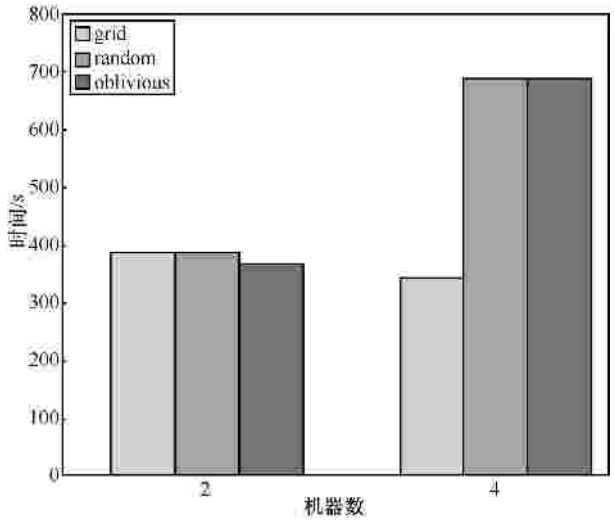

图 8 切分方式与运行时间的关系

相应增加，导致整体通信量有所增加。另外，机器数量增加后，各机器间的工作负载也可能会不均衡，导致等待时间增多。图 9（b）给出了参与计算的所有机器的内存消耗总量和与机器数量的关系。随着机器数的增加，算法消耗的内存整体上升，但是平均每台机器的消耗量有所下降。总量上升的主要原因是由于节点切分生成了多个 mirror，并且为 mirror 生成了一些附加数据结构，占用了大量内存空间。图 9（c）给出了算法在运行过程中每台机器平均通信量与机器数量的关系。可以看出曲线呈现下降趋势，这是因为随着机器增多，分配到每台机器中的顶点数目会相应减少，任意 2 台机器之间的通信量也会随着分区中顶点数目的减少而降低，所以平均通信量会减少。但集群的整体通信量有所增长，这是因为随着机器数的增加，图分区数量也相应增多，所以需要通信的节点数量也增加，导致通信总量的增加。

5 结束语

本文基于传统的集中式桥接中心度计算方法设计和实现了分布式桥接中心度计算算法 DABC。算法包含最短路径计算和点中心度计算 2 个主要的分布式过程。实验结果表明该算法比原有算法在性能上有了大幅度的提升，同时由于采用了分布式架构，该算法具备了良好的扩展性，可以支持更大规模的图数据处理。

在未来工作中，将进一步优化和完善该算法。1）设计更好的切分方法以进一步降低算法的运行时间。2）采用优化的通信方法，以降低机器间的通信开销。

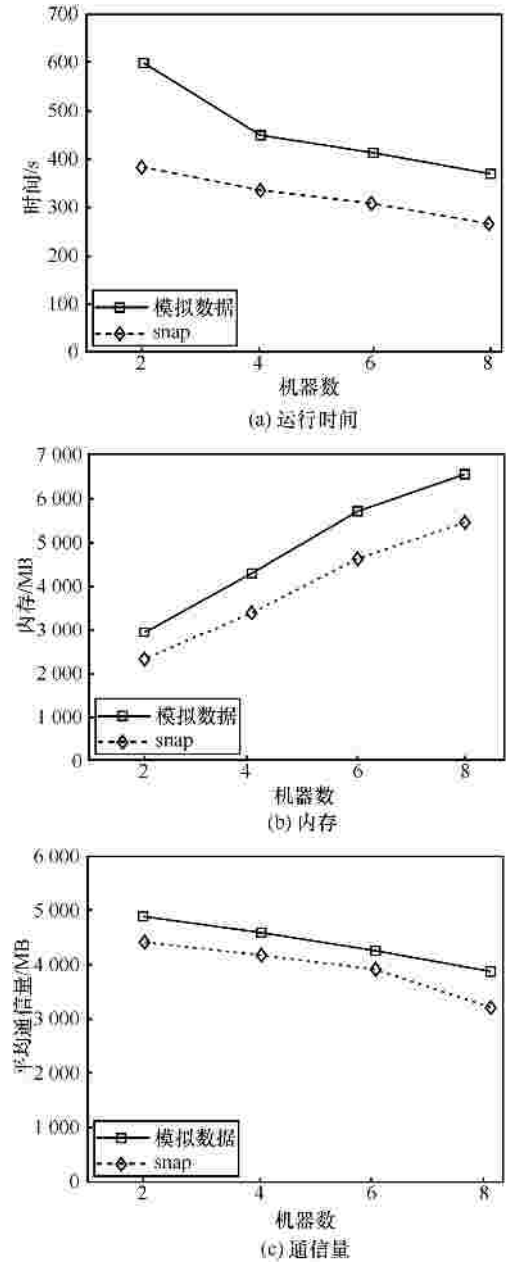

图 9 机器数与运行时间、内存、平均通信量的关系

参考文献：

[1] BADER D A, KINTALI S, MADDURI K, et al. Approximating betweenness centrality[C]//Workshop on Algorithms and Models for the Web-Graph. San Diego, CA, c2007: 124-137.

[2] ANTHONISSE J. The rush in a directed graph[M]. Amsterdam: Stichting Mathematisch Contrum, 1971. 1-10.

[3] FREEMAN L. A set of measure of centrality based on betweenness[J]. Sociomtry, 1977, 40 (1): 35-41.

[4] CARPENTER T, KARAKOSTAS G, SHALLCROSS D. Practical issues and algorithms for analyzing terrorist networks[C]// International Workshop on Mobile Commerce. San Antonio, c2012.

[5] BRANDS U. A faster algorithm for betweenness centralit Journal of Mathematical Sociology, 2001, 25 (2):163-177.

[6] BRANDS U. On variants of shortest-path betweenness centrality and their generic computation[J]. Social Networks, 2008, 30(2):136-145.
[7] LEE M, LEE J, PARK J Y, et al. QUBE: a quick algorithm for updating betweenness centrality[C]//WWW. Lyon, France, c2012: 351-360.
[8] YANG J X, WANG C K, BAI Y Y. A fast algorithm for updating betweenness centrality in social networks[J]. Journal f Computer Research and Development, 2012, 49(l): 243-249.
[9] TAN G, TU D, SUN N. A parallel algorithm for computing betweenness centrality[C]//Washington ICPP. c2009: 340-347.
[10] JEREY D, SANJAY G. MapReduce: simplied data processing on large clusters[C]//6th USENIX Symp on Operating Syst Design nd Impl. c2004: 137-150.
[11] GRZEGORZ M, MATTHEW H. Austern pregel: a system for large-scale graph processing[C]//The ACM SIGMOD Conference (SIGMOD). Indianapolis, Indiana, c2010: 135-146.
[12] LOW Y, GONZALEZ J, KYROLA A, et al. GraphLab: a new parallel framework for machine learning[C]//Uncertainty in Artificial Intelligence. c2010: 340-349.
[13] LOW Y, BICKSON D, GONZALEZ J, et al. Distributed GraphLab: a framework for machine learning and data mining in the loud[J]. Proceedings of the VLDB Endowment, 2012, 5(8): 716-727.
[14] GONZALEZ J E, LOW Y, GU H, et al. PowerGraph: distributed graphparallel computation on natural graphs[C]//USENIX Conf Operating Systems Design and Implementation. c2012: 17-30.
[15] XIN R S, GONZALEZ J E, FRANKLIN M J, et al. GraphX: a resilient distributed graph system on Spark[C]//First International Workshop on Graph Data Management Experiences and Systems (GRADES 2013). c2013: 2-16.
[16] https://giraph.apache.org[EB/OL].2011.
[17] UGANDER J, KARRER B, BACKSTROM L, et al. The anatomy o the facebook social graph[J]. arXiv preprint arXiv:1111.4503. 2011.
[18] http://graphlab.org/projects/source.html[EB/OL]. 2014.
[19] JURE L, ANDREJ K. SNAP Datasets: large network dataset collection [EB/OL]. http://snap.stanford.edu/data/ 2014.

作者简介：

高壮良（1990-），男，山东菏泽人，北京航空航天大学硕士生，主要研究方向为分布式图计算。

吕雁飞（1984-），男，辽宁朝阳人，博士，国家计算机网络应急技术处理协调中心工程师，主要研究方向为大数据技术。

张鸿（1976-），男，陕西西安人，博士，国家计算机网络应急技术处理协调中心高级工程师，主要研究方向为云计算、大数据、网络安全。