基于 MapReduce 的互联网拓扑特征参数算法研究

朱凯龙 陆余良 张岩庆
(解放军电子工程学院网络工程系 合肥 230037)

摘 要 针对传统单机算法在计算大规模互联网拓扑特征参数时效率低的问题,基于 MapReduce 分布式计算框架对网络拓扑特征参数算法进行研究。通过分析单机图算法并行移植时存在的问题,提出了图算法并行化设计的原则和消息传递机制;根据设计原则和消息传递机制,为4个网络拓扑参数设计了并行算法。实验证明,并行的拓扑参数算法能够有效提高计算效率,且具备良好的可扩展性。

关键词 互联网拓扑特征参数,MapReduce,消息传递机制,算法并行化
中图法分类号 TP301.6 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.06.013

Study on Calculation Method for Internet Topological Parameters Based on MapReduce

ZHU Kai-long LU Yu-liang ZHANG Yan-qing
(Department of Network Engineering, Electronic Engineering Institute, Hefei 230037, China)

Abstract In order to improve the efficiency of the traditional single machine algorithms in computing Internet topological parameters, we used MapReduce to study the Internet topological parameters algorithms. Through discussing the difficulties of parallelizing the traditional algorithms, this paper gave the concurrent design discipline of graph algorithms based on MapReduce and presented the message passing mechanism of graph algorithms. We designed and improved parallel algorithms to compute 4 topological parameters by using message passing mechanism. The experimental results show that the proposed MapReduce algorithm effectively improves the efficiency, and has a good scalability.

Keywords Internet topological parameters, MapReduce, Message passing mechanism, Parallel algorithm

1 引言

互联网结构复杂、规模巨大,对其网络拓扑状态和变化的研究主要从分析计算其网络特征参数入手,这些参数包括节点度数、核数、聚类系数和介数等[1-3]。目前,CAIDA 的 Skitter 项目已经通过主动探测全球路由器的方式积累了上百 TB 的拓扑数据,而且作为 Skitter 升级项目的 Ark 正在以更高的速度获取全球互联网的拓扑信息。普通单机的计算能力已经远远不能支持对如此大规模数据的分析,采用分布式计算框架,利用多个计算节点组成的集群协同完成计算任务已成为当前网络拓扑分析研究的热点。

目前,分布式处理框架主要有 MPI, Dryad, MapReduce 和 BSP。MPI[4]是早期具有代表性的并行计算模型,该模型处理效率高,可移植性强,具有很高的灵活性和很强的表达能力,但是开发者需要详细了解许多系统层面的实现细节,系统的扩展性和容错性都很难保证,不适合处理大规模数据。Dryad[5]是 Isard 等人提出的一个通用的粗粒度分布式计算模型,主要用来处理粒度较大的批量数据,该计算模型是针对运行 Windows HPC Server 的计算集群设计的,目前已经成为支撑微软云计算基础设施的重要核心技术。BSP 模型[7],即整体同步并行模型,是由著名科学家 Viliant 等人提出的一种异步的 MIMD-DM 模型,提供数据块间同步处理和数据块内异步并行的计算,被应用到 Google 的 Pregel 和 Apache 的 Hama 等项目中。MapReduce[6]由 Google 工程师 Dean 等人提出,为设计分布式算法提供了简单易懂的编程模型。著名的 Hadoop 计算平台就是 MapReduce 的一个开源实现,它凭借自身高可靠、高扩展并且开源的优越性已经成为分布式计算平台的标准框架。本文采用 MapReduce 计算框架对互联网拓扑特征参数的计算方法进行讨论。

2 MapReduce 计算框架

MapReduce 是由 Google 提出的解决分布式数据处理的计算框架。它隐藏了并行化编程繁琐的细节,将编程抽象为 Map 和 Reduce 两个过程,简化海量、分布式、高容错数据处理的开发工作,使得分布式并行处理数据变得容易。

MapReduce 的处理过程如图1所示,对大规模数据先进行分片,然后分别经过 Map, Reduce 处理后产生输出结果。Map 可以划分为 mapper, combiner 和 partitioner; Reduce 可以划分为混排、排序和 reducer。在整个过程中,数据是以键/值对的形式被处理的。

到稿日期:2016-04-06 返修日期:2016-08-27 本文受国家自然科学基金(61405248),安徽省青年科学基金(1408085QF131)资助。
朱凯龙(1991-),男,硕士生,主要研究方向为计算机网络拓扑建模、大规模图数据处理,E-mail:471801698@qq.com;陆余良(1964-),男,教授,博士生导师,主要研究方向为计算机网络安全;张岩庆(1988-),男,博士生,主要研究方向为计算机网络拓扑测量。

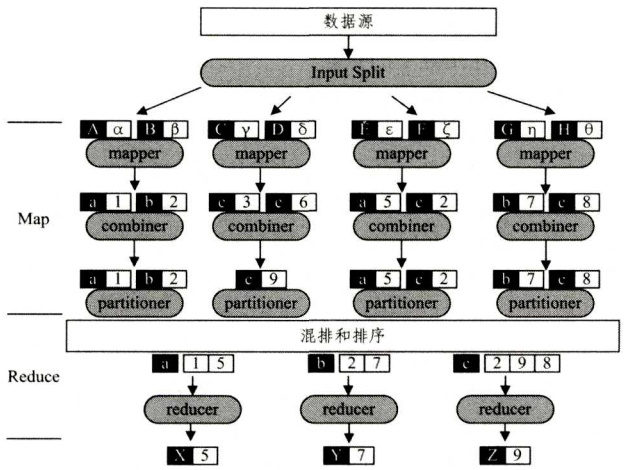

图 1　MapReduce 的处理过程

在 MapReduce 处理过程中，mapper 由用户设计编写，输入的键/值对被解析后仍然以键/值对的形式输出。combiner 是可选的本地聚合方法，对 mapper 输出进行本地聚合，减少网络数据传输。partitioner 负责将中间键/值对按照键来划分成不同的块，保证键相同的数据被分到同一个块中划分算法默认是由 MapReduce 框架提供的。混排和排序是 MapReduce 框架自动处理的，负责将 partitioner 分好的块拉取到各个 reducer 中，并按照键的大小对键/值对进行排序。reducer 由用户设计编写，对输入的键/值对进行规约处理，得到最终的结果并写入到磁盘中。

3　基于 MapReduce 的图算法设计

互联网拓扑特征参数的计算方法属于图算法范畴[8]，所以本文对并行图算法的设计进行研究。

3.1　单机图算法并行移植问题的分析

MapReduce 计算框架是运行在分布式系统中的，因为分布式系统的数据存储、处理和单机系统存在许多差异，所以传统意义上的单机算法并不能直接发挥 MapReduce 框架的并行计算能力[9]。通过分析 MapReduce 和图算法的特点，发现传统单机图算法在 MapReduce 框架中的运行存在以下问题。

(1)随机查找或修改邻居信息效率低

许多图算法都需要执行查找或修改邻居节点信息的操作，单机图算法的图结构以邻接表或者邻接矩阵的形式存储在单机内存中，都可以在常数时间内查找到邻居节点的存储位置并修改其状态信息。但是在分布式环境中，每个节点信息是以单独一条文本来存储的，查找或修改邻居节点信息都需要遍历整个图文件，效率很低。

(2)单机算法操作不具备并行特性

单机算法在设计时并没有考虑到并行特性，所以在单机上串行执行的算法并不一定能在分布式计算平台上高效地运行。如对互联网拓扑图进行遍历，在单机算法中深度优先遍历和宽度优先遍历都是常用的算法。但是同一时刻，宽度优先遍历可以并行访问处于网络同一层的多个节点；而深度优先遍历只能串行访问一个节点，在当前节点未完成访问时不能继续访问下一节点。所以，宽度优先遍历算法的并行性要优于深度优先遍历，更适合运行在 MapReduce 框架中。

(3)单机算法并行移植时会产生额外的开销

MapReduce 运行时需要执行启动作业、任务调度、读写磁盘等操作，会产生额外的时间开销。特别地，当图算法中的迭代次数过多时，每轮迭代都执行上述操作，会产生大量的额外开销，导致算法的计算效率下降。

分布式环境的数据存储、数据处理和单机系统存在很大差异，导致传统单机串行图算法向分布式并行算法移植时存在以上问题。所以本文拟重新设计适用于 MapReduce 框架的图算法，以提高互联网拓扑特征参数的计算效率。

3.2　基于 MapReduce 的图算法设计原则

针对单机图算法并行移植到 MapReduce 框架时存在的问题，本文提出在设计基于 MapReduce 的图算法时应满足以下原则。

(1)避免不同键/值对间的查找或修改操作。当算法中涉及相邻节点查找或修改操作时，可以采用主动发送消息的方法来完成，被查询的节点主动向需要该节点状态信息的节点发送消息，具体过程将在 3.3 小节中进行介绍。

(2)算法要有良好的并行特性。一次 MapReduce 任务能同时对网络中多个节点的状态进行计算。

(3)较少的 MapReduce 迭代次数。在计算量一定的情况下，提升每轮 MapReduce 处理的计算量，相应地就可以减少 MapReduce 的迭代次数。

3.3　基于 MapReduce 的图算法的消息传递机制

为避免不同键/值对间的查找或修改操作，受互联网路由报文传递的启发，本文采用消息传递机制，设计了基于MapReduce的图算法。在算法中各个节点不需要主动查找或修改邻居节点的状态信息，而是节点主动将包含更新信息的消息发送给邻居节点，邻居节点根据消息内容来更新自身节点状态，达到查找或修改邻居状态信息的目的。MapReduce 图算法中传递更新消息的过程包括 3 个步骤。

(1)产生更新消息。每个节点根据本节点的状态信息计算生成更新消息内容，把邻居节点作为消息的目的节点，并将消息发送出去。在 MapReduce 框架中，该步骤工作由 Map 阶段完成，Map 阶段的处理方法由用户完成。该阶段负责处理每条存储节点信息的文本记录，根据需求产生更新消息键/值对。其中键是邻居节点 id，值是更新消息的内容。

(2)传递消息。更新消息按照目的节点被发送给指定的节点。在 MapReduce 框架中，该步骤工作由框架的 partitioner 自动完成，它默认根据 hash 算法把具有相同键的消息键/值对和节点键/值对划分到一起，使得消息达到了传递的效果。

(3)更新节点内部状态信息。目的节点收到若干条更新消息，节点对这些消息进行解析并更新节点内部状态信息。该步骤由 MapReduce 的 Reduce 阶段完成，该阶段负责接收上一阶段传递的键/值对，把所有具有相同键的消息键/值对和节点键/值对进行聚合，得到更新后的节点键/值对并输出。Reduce 阶段的处理方法也是由用户根据需求自定义完成的。

4　基于 MapReduce 的介数算法设计

依据上述设计原则，本文设计了并行算法来计算互联网

拓扑参数，包括节点介数、聚类系数、核数和接近中心性。其中计算介数的算法最复杂，时间复杂度较高，因此以介数算法为例进行详细说明。

4.1 单机介数算法

介数[10](betweenness)常用来描述节点在网络中的重要性，反映了节点在整个网络中的作用和影响力。网络中不相邻的节点 s 和 t 之间的最短路径会经过其他若干节点，节点 v 被其他最短路径经过的次数越多，表示该节点在网络中越重要。由此，节点 v 的介数被定义为：

$$B(v)=\sum_{s\neq v\neq t\in V}\frac{\sigma_{st}(v)}{\sigma_{st}} \tag{1}$$

其中，σ_{st} 表示节点 s 和 t 之间的最短路径条数，$\sigma_{st}(v)$ 表示这些最短路径经过节点 v 的条数。使用 Floyd-Warshall 等人提出的最短路径的算法来计算介数的时间复杂度为 $O(n^3)$[11]，无法适用于大规模互联网。为此，Brandes 提出了一种快速介数算法[12]，该算法在无权无向图中的时间复杂度为 $O(nm)$，空间复杂度为 $O(m)$。

Brandes 算法的流程如图 2 所示。

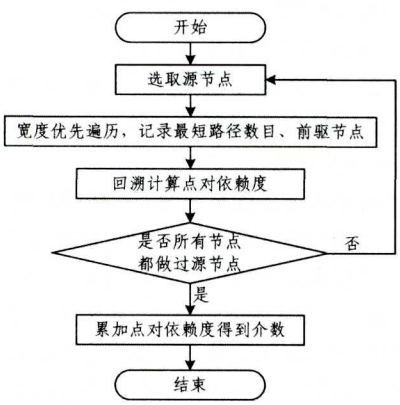

图 2 Brandes 算法流程

算法的具体步骤如下。

(1)宽度优先遍历。选取网络中一个节点 s 为源节点，宽度优先遍历其余节点，当访问到当前节点 v 时，根据式(2)计算节点 v 到源节点 s 的最短路径的数目，并记录下节点 v 的前驱节点 $P_s(v)$。迭代遍历过程，直到所有节点都被访问到。

$$\sigma_{sv}=\sum_{u\in P_s(v)}\sigma_{su} \tag{2}$$

其中，σ_{sv} 表示节点 s 到节点 v 的最短路径数目；$P_s(v)$ 表示节点 v 以节点 s 为源的前驱节点。

(2)回溯求解点对依赖度。从距离源节点 s 最远一层的节点开始回溯，按照式(3)计算当前层中节点的前驱节点的点对依赖度。不停回溯计算前驱节点的点对依赖度，直到回到源节点，最终得到节点 s 对其他所有节点的依赖度。

$$\delta_s\cdot(v)=\sum_{w:v\in P_s(w)}\frac{\sigma_{sv}}{\sigma_{sw}}(1+\delta_s\cdot(w)) \tag{3}$$

其中，$\delta_s\cdot(v)$ 表示节点 s 对节点 v 的依赖度，称为点对依赖度。

(3)变换源节点计算点对依赖度。选取其他节点作为源节点，重复步骤(1)、步骤(2)的遍历和回溯过程，直到所有节点都作过源节点。

(4)累加点对依赖度得到介数。根据式(4)，将不同源节点对节点 v 的依赖度求和，得到该节点的介数。

$$B(v)=\sum_{s\neq v\in V}\delta_s\cdot(v) \tag{4}$$

其中，$B(v)$ 表示节点 v 的介数。

4.2 基于 MapReduce 的并行介数算法

同 3.1 小节所述，并行介数算法设计存在诸多问题，本文采用基于 MapReduce 的图算法的消息传递机制解决并行移植时随机查找或修改邻居信息效率低的问题；以所有节点为源，并发进行遍历和回溯过程来解决算法迭代次数多的问题，提高算法的并行化程度，减少启动作业、读写磁盘的额外开销。

并行介数算法的流程如图 3 所示，所有节点以自己为源节点并发地进行宽度遍历，当所有源节点完成宽度遍历后进入回溯阶段，回溯阶段仍然是所有源节点并发执行。回溯阶段完成后便获得所有点对之间的依赖度，最终累加点对依赖度即可得到每个节点的介数。其中，遍历和回溯阶段是算法的核心步骤，这两个过程均采用了图算法的消息传递机制，下文详细介绍这两个步骤。

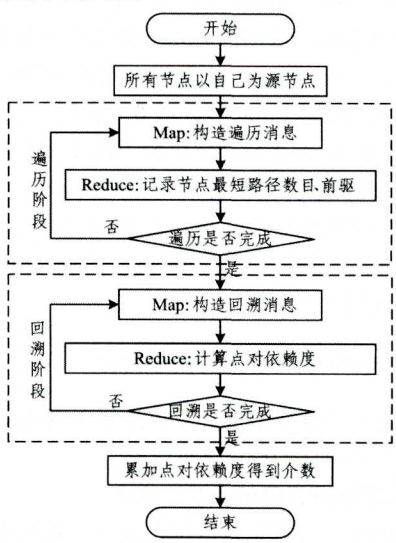

图 3 并行的介数算法

算法遍历阶段的消息传递过程如图 4 所示。

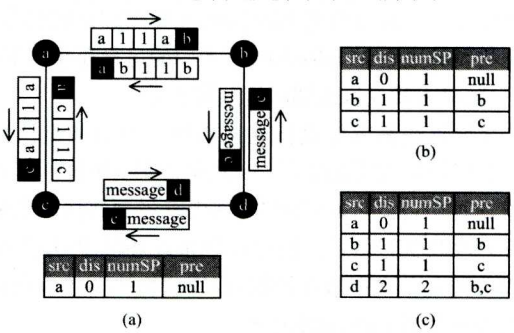

图 4 算法遍历阶段的消息传递过程

在该过程中，网络中每个节点维护着一个状态记录表，表中每条记录包含当前已经访问到的源节点 id 和相应的距离、最短路径条数和前驱节点。节点 a 初始状态的记录表如图 4(a)所示。第一轮迭代时，所有节点需要向邻居节点发送更新消息，同时需要接收邻居发来的更新消息，根据更新消息修改自身节点的记录表。经过第一轮迭代后，所有节点的记录表都会被更新，得到所有源节点距离为 1 的遍历结果，此时节点 a 的记录表如图 4(b)所示。同理进行第二轮迭代，得到所有

源节点距离为 2 的遍历结果，节点 a 的记录表如图 4(c)所示。继续迭代，直到所有源节点都完成遍历，每个节点的记录表中都记录着到所有源节点的信息，算法遍历阶段完成。

算法回溯阶段与遍历阶段的过程类似，以所有节点为源节点，从距离最远一层的节点开始回溯，计算源节点对当前节点的前驱节点的依赖度。不停回溯，直到回溯到所有源节点为止，每个节点记录表中都记录着所有源节点对该节点的依赖度，回溯阶段完成。宽度遍历和回溯的 MapReduce 伪代码如算法 1 所示。

算法 1　并行介数算法的核心步骤

宽度优先遍历阶段

```
1. class BFSMapper
2.   method Map(id n,node N )
3.     Emit(id n,node N )
4.     for all m∈N. neighbor do
5.       for all record∈N. records do
6.         if record. distance=round
7.           message. source ←record. source
8.           message. distance ←record. distance+1
9.           message. numSP ←record. numSP
10.          message. predecessor←n
11.    Emit(id m,Message message)

1. class BFSReducer
2.   method Reduce(id m,[p1 ,p2…])
3.     find node M from [p1 ,p2…]
4.     for all p∈[p1 ,p2…]−M do
5.       for all record∈M. records do
6.         if p. source=record. source
7.           record. distance ←message. distance
8.           record. numSP ←message. numSP
9.           record. predecessors. add(p. predecessor)
10.    Emit(id m,node M)
```

回溯求解点对依赖度阶段

```
1. class BackTrackMapper
2.   method Map(id n,node N)
3.     Emit(id n,node N)
4.     for all m∈N. predecessors do
5.       for all record∈N. records do
6.         if record. distance=round
7.           message. source ←record. source
8.           message. numSP ←record. numSP
9.           message. dependency←record. dependency
10.    Emit(id m,Message message)

1. class BackTrackReducer
2.   method Reduce(id m,[p1 ,p2…])
3.     find node M from [p1 ,p2…]
4.       for all p∈[p1 ,p2…]−M do
5.         for all record∈M. records do
6.           if record. source=p. source
7.             record. dependency←M. dependency+
8.             M. numSP/p. numSP * (1+p. dependency)
9.     Emit(id m,node M)
```

并行介数算法的遍历阶段和回溯阶段都包含了 Map 和 Reduce 过程，分别通过发送更新消息实现对网络节点信息修改的目的。需要指出的是，如宽度优先遍历阶段中 BFSMapper 第 3 行所示，两个阶段的 Map 在发送更新消息的同时还发送了节点的拓扑结构信息，这是为了传递网络的拓扑结构，使得算法可以继续进行迭代。

该并行介数算法不需要每次选定一个源节点计算依赖度，而是所有节点作为源节点并发地进行遍历和回溯过程，每轮 MapReduce 迭代可以更新关于所有源节点的信息。整个算法只需进行一次遍历阶段和回溯阶段即可求出所有的点对依赖度，大大减少了算法的 MapReduce 迭代次数。

并行介数算法需要处理 $O(dn^2)+O(mn)$ 对键/值对，因为互联网的小世界特性[13]，所以 d 为常数，因此需要处理的键值对数为 $O(mn)$。算法运行在包含 N 个计算节点的集群中的时间复杂为 $O(mn/N)$。计算过程需要存储任意节点对的最短路径数目，节点对数目为 $O(n(n-1)/2)$，所以并行介数算法的空间复杂度为 $O(n^2)$。

5　算法分析与实验

5.1　实验环境

本文采用了运行在 Hadoop 平台的 MapReduce 框架进行实验。选用的 Hadoop 版本为 1.2.1，在以太网环境下搭建计算集群，集群拓扑结构如图 5 所示。

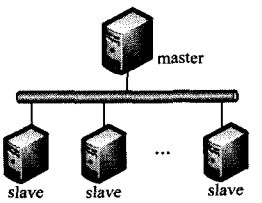

图 5　Hadoop 计算集群的拓扑结构

实验搭建的 Hadoop 计算集群包括 1 个 master 节点和 8 个 slave 节点，集群节点的详细配置如表 1 所列。

表 1　集群节点配置信息

集群节点	CPU	内存	硬盘	网卡
master	i7-4770 (4 核 3.4GHz)	16GB	500G/7200RPM	1000M
slave	i7-4770 (4 核 3.4GHz)	4GB	1T/7200RPM	1000M

5.2　实验数据

实验数据来自于 CAIDA 的 Ark 项目，Ark 项目通过部署在全球不同地区的 132 个探测源对全球 IP 集进行主动探测来获取 traceroute 数据。本文根据 2015 年的探测数据构建出路由器网络，选取了 5 个不同地区的路由器网络拓扑 G_1-G_5。为了比较实验，选取了不同规模的网络，网络规模如表 2 所列。

表 2　2015 年 5 个地区的网络规模

编号	地区	n	m	$\langle k\rangle$
G_1	IN	280759	281349	2.01
G_2	JP	774619	789841	2.00
G_3	DE	1847669	1863544	2.02
G_4	KR	2035168	2046123	2.01
G_5	US	6198093	6322708	2.04

其中，n，m 表示网络的节点数和边数，$\langle k\rangle$ 表示网络的节点平均度。

5.3 实验结果

除并行介数算法外，本文基于 MapReduce 计算框架设计并实现了并行的聚类系数、核数和接近度中心性算法，将 4 种算法在不同规模的 Hadoop 集群、不同大小的网络数据上进行对比实验。

并行介数算法在不同规模的集群、不同数据上运行，结果如图 6 所示。可以看出，随着集群中计算节点的增加，计算介数的时间在不断下降，并且随着实验网络规模的增大，运行时间减少的幅度增大。实验说明基于 MapReduce 的并行介数算法可以通过扩大集群规模来大幅提升效率，尤其是在处理大规模拓扑数据时更具优势。使用 8 个计算节点来计算包含百万级别节点的网络，效率可以提升 6 倍以上。

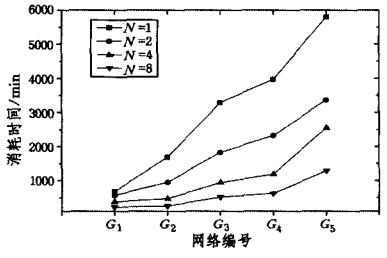

图 6 并行介数算法的消耗时间

利用基于 MapReduce 的并行算法计算实验网络的拓扑特征参数，5 个地区网络的拓扑特征参数如表 3 所列。其中 *A_BC* 表示节点平均介数，*A_Cluster* 表示节点的平均聚类系数，*Core* 表示网络的核数，*A_Close* 表示节点的平均接近中心性。

表 3 5 个地区网络的拓扑特征参数

编号	*A_BC*	*A_Cluster*	*Core*	*A_Close*
G_1	0.0135	0.0023	5	0.1467
G_2	0.0149	0.0045	7	0.1492
G_3	0.0133	0.0021	15	0.1533
G_4	0.0128	0.0022	10	0.1541
G_5	0.0152	0.0072	18	0.1604

使用并行算法加速比[14] *SP* 来比较并行算法与单机算法的计算效率，加速比计算方法如式(5)所示。

$$SP(N)=\frac{T(1)}{T(N)} \tag{5}$$

其中，$T(N)$表示算法在 N 台计算机上运行的时间。

本文设计实现的并行算法的加速比如图 7 所示。可以看出，随着计算集群规模的扩大，算法效率提升越来越明显。当计算节点为 2,4,8 时，4 种算法的平均加速比分别为 1.60，2.46，4.77。实验说明了并行互联网拓扑特征参数算法在分布式计算平台上具有良好的扩展性，与单机算法相比在效率上具有巨大的优势。

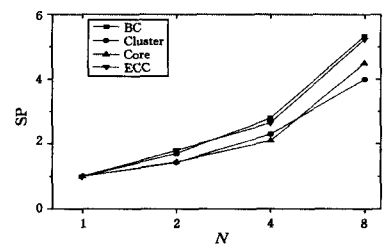

图 7 并行算法的加速比

结束语 本文使用 MapReduce 分布式计算框架对互联网特征参数进行计算，为互联网拓扑特征参数设计了并行算法，提高了网络拓扑特征参数的计算效率，有助于进一步挖掘大规模互联网的拓扑特性，为互联网拓扑的态势感知、演化模型的建立和异常检测等工作提供支持。分布式计算技术发展迅速，MapReduce 计算框架仍然存在一定的不足，在下一步研究中，可以通过改进 MapReduce 计算框架或选用更合适图运算的计算框架来完成互联网拓扑特征参数的计算，进一步提高特征参数的计算效率。

参 考 文 献

[1] 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.

[2] ZENG W, XU M W, WU J P. Survey of Network Topology Models [J]. Application Research of Computers, 2005, 22(7): 1-8. (in Chinese)
曾伟，徐明伟，吴建平. 网络拓扑模型评估述评[J]. 计算机应用研究，2005，22(7)：1-8.

[3] 郭世泽，陆哲明. 复杂网络基础理论[M]. 北京：科学出版社，2012：40-43.

[4] FOSTER I, KESSELMAN C. The Grid 2 Blueprint for a new computing infrastructure(second edition)[M]. Burlinton: Morgan Kaufman, 2003.

[5] ISARD M, BUDIU M, YU Y, et al. Dryad: distributed data-parallel programs from sequential building blocks[J]. ACM Sigops Operating Systems Review, 2007, 41(3): 59-72.

[6] DEAN J, GHEMAWAT S. MapReduce: Simplified data processing on large clusters [J]. Communication of the ACM, 2008, 51(1): 107-113.

[7] VALIANT L G. A bridging model for parallel computation [J]. Communications of the ACM, 1990, 33(8): 154-166.

[8] 谷峪，于戈，鲍玉斌. 大规模图数据的分布式处理[M]. 北京：清华大学出版社，2015：23-27.

[9] JIMMY L, MICHAEL S. Design patterns for efficient graph algorithm in MapReduce [C]// Eighth Workshop on Mining and Learning with Graphs 2010. Washington, DC, 2010.

[10] FREEMAN L C. Centrality in social networks: Conceptual clarification [J]. Social Networks, 2012, 1(3): 215-239.

[11] PROUNTZOS D, PINGALI K. Betweenness centrality: algorithms and implementations [J]. ACM Sigplan Notices, 2013, 48(8): 35-46.

[12] BRANDES U. A faster algorithm for betweenness centrality [J]. Journal of Mathematical Sociology, 2010, 25(2): 163-177.

[13] 汪小帆，李翔，陈关荣. 复杂网络理论及其应用[M]. 北京：清华大学出版社，2006：49-55.

[14] XIE C, MAI L D, DU Z H, et al. Research and analysis of Parallel computing system speedup[J]. Computer Engineering and Applications, 2003, 39(26): 66-68. (in Chinese)
谢超，麦联叨，都志辉，等. 关于并行计算系统中加速比的研究与分析[J]. 计算机工程与应用，2003，39(26)：66-68.