中国科学: 信息科学 2018 年 第 48 卷 第 7 期: 932–946
SCIENTIA SINICA Informationis
复杂系统智能仿真与控制专刊 · 论文

基于 Pregel 的大规模网络传播仿真算法设计及实现

艾川[1], 陈彬[1*], 刘亮[1], 董健[1], 何凌南[2], 赖凯声[2], 邱晓刚[1]

1. 国防科技大学系统工程学院, 长沙 410073
2. 中山大学传播与设计学院, 广州 510006
* 通信作者. E-mail: nudtcb9372@gmail.com

收稿日期: 2018–03–27; 接受日期: 2018–04–08; 网络出版日期: 2018–07–18

国家重点研究发展计划 (批准号: 2017YFC1200300)、国家自然科学基金 (批准号: 71673292, 61503402, 71673294)、国家社会科学基金 (批准号: 17CGL047) 和广东省舆情大数据分析与仿真重点实验室资助项目

摘要 随着互联网和在线社交媒体的快速发展, 社交网络中的信息传播十分迅速并难以掌控, 社交网络中信息传播的规律研究需要进行大量的网络传播计算实验. 当前以 SIR 模型为基础的网络传播实验广泛应用于疾病传播研究和信息传播研究. 目前受限于硬件、软件等原因还很难进行超大规模的网络传播计算. 然而当前互联网信息传播呈现出用户规模大、信息多、传播速度快的特点, 以往基于抽象简化方法设计的小规模网络传播实验渐渐难以满足日益增长的计算需求. 本文借助 Spark 平台, 实现了超大规模网络传播计算实验算法, 将该算法与 Nepidemix 单机传播计算组件进行了性能对比, 论证了其优势和不足. 该算法在有足够集群计算资源时可以进行上亿规模节点的网络传播实验, 并且开发难度小, 基于此可以高效地进行大规模复杂网络中的信息传播仿真, 为舆情传播、监测、预测和干预等研究奠定了基础.

关键词 信息传播, 分布式计算, SIR 模型, 复杂网络, 仿真方法

1 引言

许多现实系统的拓扑结构都可以用复杂网络表示. 比如在研究疾病传播的过程中, 可以用网络的节点代表个体, 边代表个体之间的接触或者好友关系. 1998 年 Watts 等 [1] 提出小世界 (WS) 网络模型, 1999 年 Barabási 等 [2] 提出无标度网络模型, 并由其他研究者加以扩充, 这些模型在一定程度上反映了显示网络的自组织形成机制, 极大地促进了复杂网络的研究.

众多研究以复杂网络为基础, 构建 SIR (susceptible infected recovered model) 等疾病传播模型, 通过仿真等方法研究疾病传播的规律和管控方法. 如文献 [3∼5] 用平均场方法分别研究了节点无线的无标度网络中 SIS (susceptible infected susceptible model) 和 SIR 模型, 发现了与经典流行病阈值理论有

引用格式: 艾川, 陈彬, 刘亮, 等. 基于 Pregel 的大规模网络传播仿真算法设计及实现. 中国科学: 信息科学, 2018, 48: 932–946, doi: 10.1360/N112017-00302
Ai C, Chen B, Liu L, et al. Design and implementation of large-scale network propagation simulation method inspired by Pregel mechanism (in Chinese). Sci Sin Inform, 2018, 48: 932–946, doi: 10.1360/N112017-00302

本质差异的传播阈值为零的特性. 目前也有大量研究基于随机模型对网络中信息传播进行仿真, 比如线性阈值模型 [6~8]、独立级联模型 [9,10], 以及流行病模型 [3,11,12] 等.

可以看到, 复杂网络中的传播计算是疾病和信息传播相关研究的重要组成部分. 部分常见的研究复杂系统建模的工具包括元胞自动机、Netlogo, 以及一些多智能体建模方法. 元胞自动机是一类模型的总称, 也是一个方法框架. 其特点是时间、空间、状态都离散, 每个变量只取有限多个状态, 且其状态改变的规则在时间和空间上都是局部的. NetLogo [13] 是用于模拟自然和社会现象的编程语言和建模平台, 特别适合于模拟随时间发展的复杂系统. Zhang 等 [14] 将认知建模, Agent 组织理论和基于 DEVS 的框架结合在一起, 实现了大规模系统的一种新的多智能体建模方法. 类似的建模方法使用的平台有 Repast HPC [15], DSOL [16] 等. 这类建模方法适合于对一般复杂系统进行建模, 但没有对图 (网络) 相关算法进行深入研究和优化, 而复杂网络本身具有区别于一般复杂系统的特性, 比如小世界特性、无标度特性等, 同时复杂网络中的传播计算有如节点的度、聚类系数、社团结构等图相关的特殊的快速计算需求. 因此这类建模方法并不适用于大规模复杂网络传播计算, 大规模复杂网络传播计算需要更专业的工具.

常用的专门用于复杂网络计算的工具有很多, 如 NetworkX, Networkit, SNAP 等, Staudt 等 [17] 做了详细的对比, 各工具在复杂网络计算上各有优势, 研究者需要根据需求选用合适的工具. 如 Gephi [18] 适合用来做复杂网络可视化的研究, Pajeck [19] 类似于 Networkit, 有一定复杂网络计算和可视化的能力. SNAP (Stanford network analysis project) [20] 采用了 C++ 和 Python 混合接口的模式, 是一个通用网络分析和图挖掘库, 也有一定的大规模复杂网络计算能力. KDT [21], GraphCT [22], STINGER [23] 和 Ligra [24] 通过在本地实现并行的内核提供高性能的网络分析. 近年来, 基于 Python 语言的软件包 Nepidemix [25] 越来越受到重视, 它具有较强的复杂网络分析处理能力、可扩展性强、开发成本低, 对研究者比较友好, 在非超大规模的网络分析中性能上满足一般需求, 与其他类似组件差别较小, 可以满足日常使用. 但是大规模网络的相关计算 [17] 仍然需要专业的并行大规模图计算组件, 如 GraphLab 和 Pregel 等.

随着社交网络以及自媒体的迅猛发展, 截至 2017 年 6 月, 中国网民规模达到 7.51 亿[1)], 大量网民积极参与网上互动, 每一个个体都有可能成为传播的热点和中心. 大规模复杂网络传播仿真迫切需要新的方法和工具. 在 Google 搜索引擎平台上, 每月有 40 亿小时的视频、4.25 亿 Gmail 用户和 1.5 亿 GB 网页索引, 却能实现 0.25 秒搜索出结果. 其中最为关键的就是 Google 提出的 Pregel 算法框架, 该算法用于支持 PageRank 算法计算搜索结果时效率极高, 显示了这类并行大规模图 (网络) 计算组件的优势.

本文基于 Spark 平台中 GraphX 组件, 使用其中的 Pregel API, 基于 Google 的 Pregel 框架原理, 设计并实现了大规模复杂网络信息传播计算方法; 设计实验通过使用该方法和 Nepidemix 组件运行相同的模型, 对该方法的性能进行检测. 第 2 节介绍了复杂网络传播仿真常用的两个模型和本文使用的图计算组件, 并对相关计算平台进行了对比; 第 3 节提出了基于图迭代的传播仿真方法; 第 4 节通过实验检验了该方法的性能, 并对模型和实验进行讨论; 第 5 节对文章内容进行总结.

2 相关工作

2.1 传播模型

信息传播与疾病传播有极大的相似性, 关于传染病传播的数学模型研究是从 Enko (1889) 开始的,

1) http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/hlwtjbg/201701/P020170123364672657408.pdf.

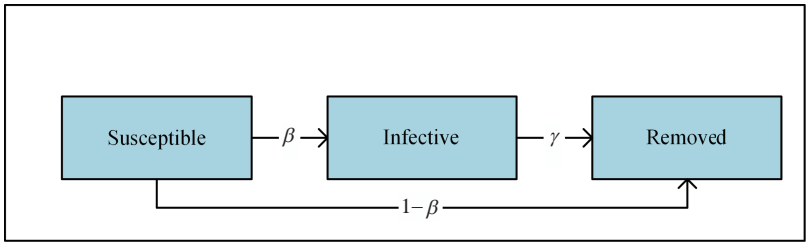

图 1 (网络版彩图) SIR 模型状态转移图

Figure 1 (Color online) State transition of SIR model

1927 年 Kermarck 和 Mckendrick [26] 利用动力学的方法建立的 SIR 传染病模型成为该领域内奠基性的工作, 并对其传播规律和流行趋势进行了研究. 在传染病模型里, 一般把总人口 N 分为易感者类 S, 染病者类 I 和恢复者类 R. 一个 SIRS 类模型, 它表示易感者被染病者传染成为染病者个体, 染病者具有免疫后从感染者类移出变为恢复者. 恢复者渐渐失去免疫力后又变为易感者类. 假设 t 时刻易感者、染病者和恢复者的数量分别为 $S(t)$, $I(t)$ 和 $R(t)$, 则三者之和等于 $N(t)$, 即总的人口数.

SIR 模型的建立基于以下 3 个假设:

(1) 不考虑人口的出生、死亡、流动等种群动力因素. 人口始终保持一个常数, 即 $N(t) \equiv K$.

(2) 一个病人一旦与易感者接触就必然具有一定的传染力. 假设 t 时刻单位时间内, 一个病人能传染的易感者数目与此环境内易感者总数 $S(t)$ 成正比, 比例系数为 β, 从而在 t 时刻单位时间内被所有病人传染的人数为 $\beta S(t)I(t)$.

(3) t 时刻, 单位时间内从染病者中移出的人数与病人数量成正比, 比例系数为 γ, 单位时间内移出者的数量为 $\gamma I(t)$.

$$\begin{cases} \mathrm{d}I/\mathrm{d}t = \beta SI - \gamma I, \\ \mathrm{d}S/\mathrm{d}t = -\beta SI, \\ \mathrm{d}R/\mathrm{d}t = \gamma I. \end{cases} \tag{1}$$

有 $I = (S_0 + I_0) - S + \frac{1}{\sigma}\ln\frac{S}{S_0}$, 其中, σ 是传染期的接触数, $\sigma = \frac{\beta}{\gamma}$. 状态转换图如图 1 所示.

SIR 模型适用于康复后获得终生免疫的传染病, 该模型虽然简单, 但得到了非常有价值的概念和结论. 流行病的传播很难通过实际的实验进行验证, 通过构建模型来进行仿真实验可以对疾病传播的机理和管控方法进行研究. 但是上述 SIR 模型并没有考虑人口动力学过程, 有些传染病流行期比较长, 有可能导致人口总数的明显变化, 这样的过程中人口动力学过程不能忽略.

为了刻画信息在社交网络环境中的传播过程, Liu 等 [27] 提出了 SVFR (susceptible, view, forward, removed) 模型, 该模型是基于 SIR 等疾病传染模型改进得来的. 但一个用户浏览朋友信息的过程可能是非线性非同质化的, 对各个状态的定义也是不同的, 因此提出了更符合信息传播实际特征的 SVFR 模型.

在 SVFR 模型中, 每个节点都可以是以下 4 个状态的任何一个.

- Susceptible (S): 信息的潜在阅读者.
- View (V): 查看该信息的用户.
- Forward (F): 转发该信息的用户.
- Removed (R): 忽略信息的用户. 可能是对该信息不感兴趣或者已经看过这个信息的用户.

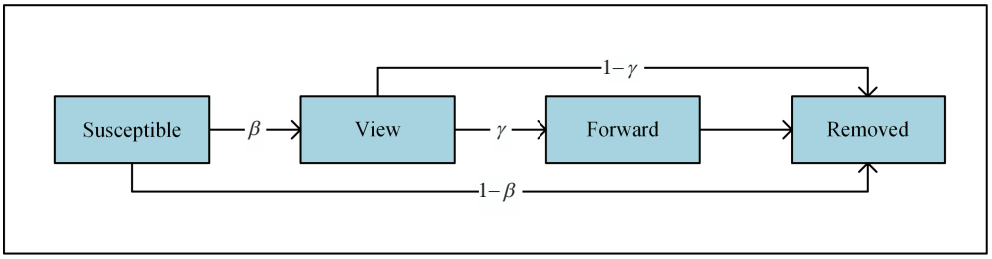

图 2 (网络版彩图) SVFR 模型状态转移图

Figure 2 (Color online) State transition of SVFR model

在该模型中, 在第 0 步时, 有一个节点被初始化为转发该信息的用户 Forward (F), 其余的所有的节点都被初始化为 Susceptible (S). 状态转换如图 2 所示.

采用该模型的系统中, 信息的潜在阅读者 Susceptible 如果接收到该信息, 将以 β 的概率阅读该信息, 状态改变为 View, 以 $1-\beta$ 的概率选择不阅读, 并将状态改变为 Removal, 如果查看信息将以 γ 的概率转发该信息状态变为 Forward, 以 $1-\gamma$ 的概率选择不转发并改变状态为 Removed. 到达转发 Forward 状态的节点, 将在下一个步骤中状态改变为 Removed.

本文采用了 SVFR 模型作为信息传播模型来测试大规模网络传播算法的性能, 可采用的模型有很多, 例如 SIR, SIS 等模型都是比较经典的模型, 但是就信息传播领域, SVFR 模型是切合我们信息传播仿真这一研究目标的模型. 就计算性能而言, 采用 SIR 或者 SVFR 模型并不影响大规模复杂网络计算组件与一般工具的对比. 因此, 采用该模型去进行网络中的信息传播实验是合理的.

2.2 并行图算法 Pregel

一般而言, 并行图计算会遇到很多问题, 难点在于合理地进行数据的分布式存储和并行图计算. 当前有几个典型的大规模网络计算平台, 如 Google 开发的 Pregel 计算框架 [28~30], GraphLab, 以及 Spark 中的 GraphX 组件.

Google 公司为了提高搜索排序算法性能, 根据 BSP (bulk synchronous parallel) 原理设计了 Pregel [31] 框架. Pregel 是一个用于分布式图计算的计算框架, 主要用于图遍历 (bridth-first search, BFS)、最短路径 (shortest path problem, SSSP)、PageRank 计算等等. 由于分布式的计算环境 Hadoop 下 MapReduce 进行图计算效率很低, 并行图算法库 Parallel BGL 或者 CGMgraph 没有容错机制, Google 开发了这个新的计算框架.

GraphLab [32,33] 是由 CMU (卡内基梅隆大学) 的 Select 实验室在 2010 年提出的一个基于图像处理模型的开源图计算框架. 该框架最初是为处理大规模机器学习任务而开发的, 但是该框架也同样适用于许多数据挖掘方面的计算任务. 在并行图计算领域, 该框架在性能上高出很多其他并行计算框架 (例如, MapReduce, Mahout) 几个数量级.

Spark 是一个开源框架 [34], 源自加州大学伯克利分校 (University of California, Berkeley) 的 AMPLab, 它继承了 MapReduce 的线性扩展性和容错性, 同时对它做了一些非常重要的拓展. 更为重要的是, Spark 为数据科学分析人员提高了生产率, 它将数据预处理、建模分析、模型评价等一系列过程整合在一个编程环境中, 大大加速了开发过程. Spark 还紧密集成了 Hadoop 生态系统里的许多工具. 它能够读写 MapReduce 支持的所有数据格式, 还可以与 Hadoop 上常用的数据格式, 如 AVRO, PARQUET 和 CSV 等进行交互. 还能与 NoSQL 数据库如 HBase 进行交互. 它可以运行在 Hadoop

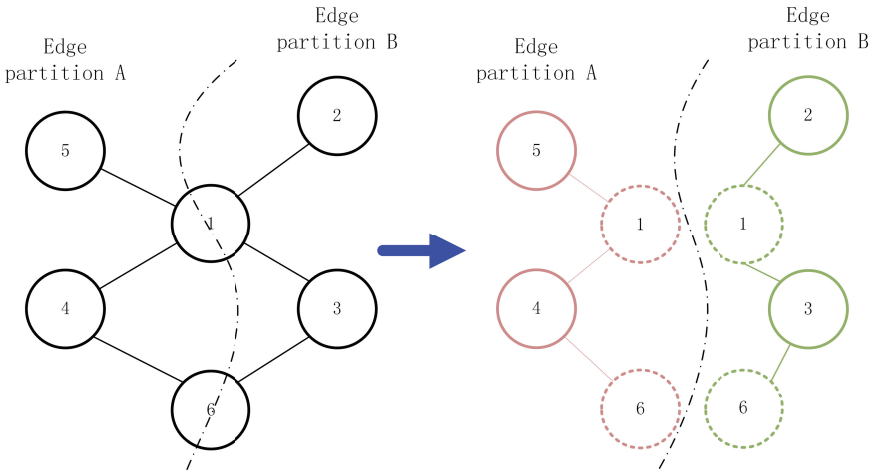

图 3　(网络版彩图) GraphX 图划分方法

Figure 3　(Color online) Graph partition of GraphX

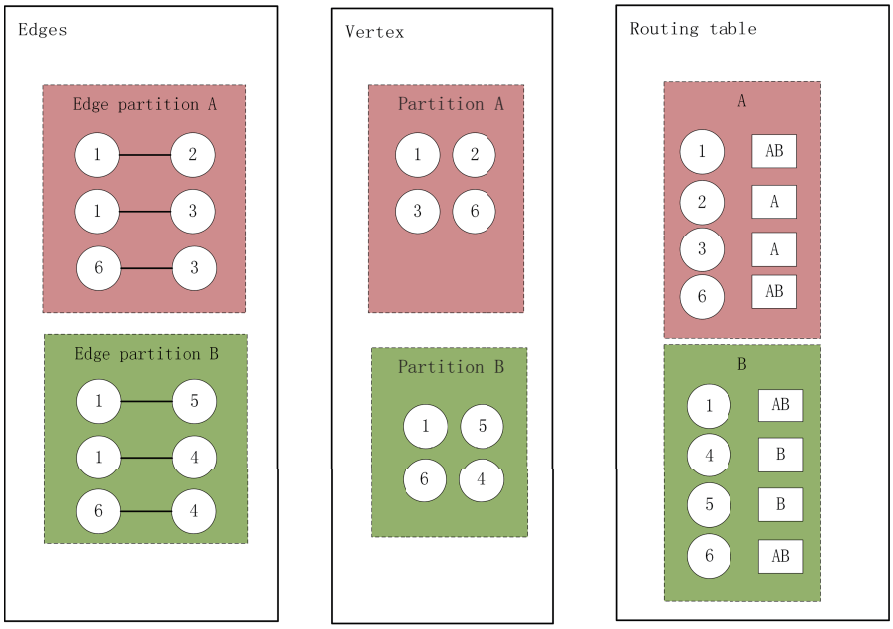

图 4　(网络版彩图) GraphX 图计算数据结构

Figure 4　(Color online) Data structure of GraphX

集群调度和资源管理器 YARN 上, 如此, Spark 可以和 MapReduce 和 Impala 等其他处理引擎动态共享集群资源和管理策略.

GraphX 是 Spark 中的图并行计算组件. 它支持 Pregel, Giraph 和 GraphLab 中的许多图并行处理任务. 由于 GraphX 基于 Spark, 它在进行以超大规模的网络为中心的分析计算时是非常方便的. 特别是 GraphX 点切的图划分方法和策略, 如图 3 和 4 为图切分存储以及后续的图计算带来了极大的便利. 其特点就是图切分采用点切的方法, 数据中既存了节点属性, 还存了边的属性, 并且节点和边分别用 Spark 独有的弹性分布式数据集 RDD 来存储, 计算时效率非常高.

GraphX 和 GraphLab 开发者的 Conzalez [32,33] 做了很多关于 GraphLab, GraphX 和 Google 的 Pregle 项目的性能测试, 发现基于 GraphLab 开发的 PowerGraph 和 GraphX 更适合做符合幂律分布的计算, 即能够有效处理网络中部分节点度非常大的情况, Pregel 和 GrpahLab 都很难处理. PowerGraph 的性能比 GraphX 快两倍左右, 其主要原因是 PowerGraph 是由 C++ 实现的, GraphX 是 scala 实现的, Java 开销比较大.

这几个大规模图计算平台中, Google 的 Pregel 平台并没有开放使用, 我们很难采用. GraphLab 和 Spark 中 GraphX 相比而言图计算方面的优势较大, 但是 GraphX 的平台 Spark 在数据处理方面有非常显著的优势, 对于网络数据处理、网络提取和进一步分析等整体流程而言可以提高效率. 同时需注意, GraphLab 由于接受了融资已经变成了收费项目, 没有相关支持的情况下不适合用来开展研究. 基于 Google Pregel 的原理, AMPLab 在开发 Spark 时向图组件模块 GraphX 中加入了 Pregel API. 这个 API 是 Spark 中的迭代应用 aggregateMessage 的一个典型案例, 用它可以在图中方便地迭代计算, 如最短路径、关键路径、N 度关系等.

Pregel API 中的计算分为一个个超步 (superstep), 这些 superstep 中执行流程如下:

- 首先输入图数据, 并进行初始化.
- 将每个节点均设置为活跃状态. 每个节点根据预先定义好的 sendmessage 函数, 以及方向 (边的正向、反向或者双向) 向周围的节点发送信息.
- 每个节点接收信息如果发现需要计算则根据预先定义好的计算函数对接收到的信息进行处理, 这个过程可能会更新自己的信息. 如果接收到消息但是不需要计算则将自己状态设置为不活跃.
- 每个活跃节点按照 sendmessage 函数向周围节点发送消息.
- 下一个 superstep 开始, 像步骤 3 一样继续计算, 直到所有节点都变成不活跃状态, 整个计算过程结束.

以一个具体例子来说明这个过程: 如图 5 所示, 假设一个图中有 5 个节点, 从左到右依次为第 1, 2, 3, 4, 5 个节点. 节点的状态转移过程如图 6. 圈中的数字为节点的属性值, 实线代表节点之间的边, 虚线是不同超步之间的信息发送, 带阴影的圈是不活跃的节点. 我们的目的是让图中所有节点的属性值都变成最大的那个属性值.

- Superstep 0: 首先所有节点设置为活跃, 并且沿正向边向相邻节点发送自身的属性值.
- Superstep 1: 所有节点接收到信息, 节点 1 和 5 发现自己接受到的值比自己的大, 所以更新自己的节点 (这个过程可以看做是计算), 并保持活跃. 节点 2, 3, 4 没有接收到比自己大的值, 所以不计算、不更新, 变为不活跃. 活跃节点继续向相邻节点发送当前自己的属性值.
- Superstep 2: 节点 2 接受信息并计算, 其他节点没接收到信息, 所以接下来只有节点 2 活跃并发送消息.
- Superstep 3: 节点 3 接受信息并计算, 其他节点没接收到信息, 所以接下来只有节点 3 活跃并发送消息.
- Superstep 4: 节点 4 接受信息不计算变为不活跃, 所有节点不活跃, 算法终止.

在 Pregel API 计算框架中, 信息发送函数和节点计算函数是两个核心函数. 该方法采用了整体并行同步计算框架 (BSP) [35], 通常而言, 整体并行同步计算框架需要 3 个部分:

- 能够处理本地存储器事务的组件 (即处理器);
- 在这些组件对之间传递消息的网络;
- 一个允许所有或部分组件的同步硬件设施.

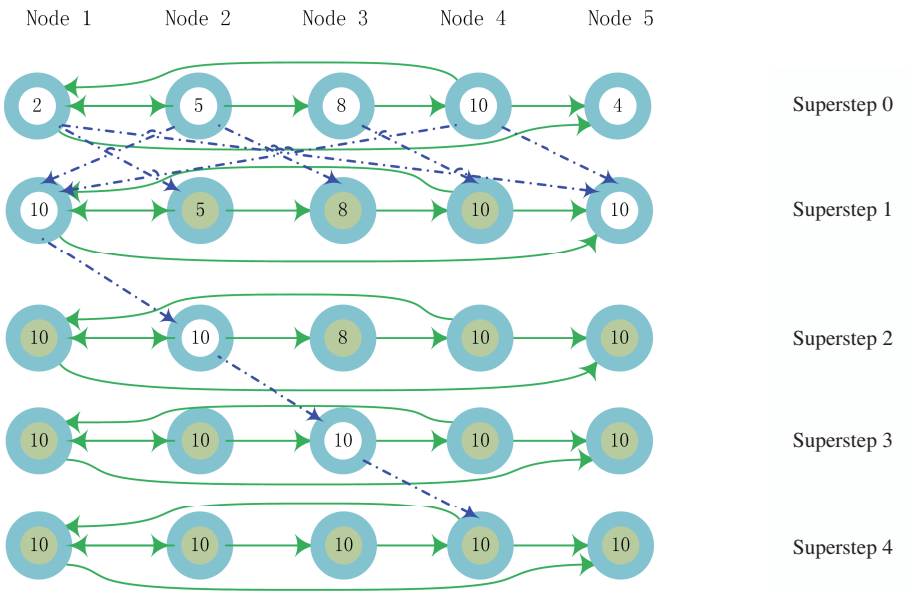

图 5 (网络版彩图) 用 Pregel 寻找最大值, 其中阴影的节点是不活跃的

Figure 5 (Color online) Find max value by Pregel, shadow node is inactive

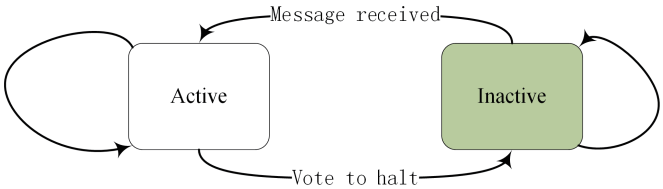

图 6 (网络版彩图) 节点的状态转移

Figure 6 (Color online) State transition

这通常被解释为可以遵循不同的计算线程的一组处理器, 每个处理器配备有快速本地存储器并且通过通信网络互连. BSP 算法由一系列的 superstep 组成, 每个 superstep 由 3 部分组成.

- 并发计算: 每个参与的处理器可以执行本地计算, 即每个进程只能使用存储在处理器的本地快速存储器中的值. 计算异步发生, 但可能与通信同时进行.
- 通信: 进程之间交换数据.
- 栅栏同步: 当一个进程到达这个点 (栅栏), 它等待, 直到所有其他进程已经到达相同的栅栏.

3 基于图迭代的传播仿真算法

本节介绍如何基于 GraphX 中的 Pregel API 设计基于图迭代的传播仿真算法. 详细阐述如何基于 Pregel 实现 SVFR 模型, 包括信息和节点属性定义、发消息函数定义以及收消息函数定义. Pregel API 调用方法:

$$\text{Pregel}(g, \text{initialMsg}, \text{maxIterations}, \text{edgeDirection})(\text{vprogFunc}, \text{sendMsgFunc}, \text{mergeMsgFunc}).$$

各参数含义如下所述.

- g: 初始网络.
- initialMsg: 初始信息.
- maxIterations: 最大迭代次数.
- edgeDirection: 边方向, 即向一条边的哪边发信息.
- vprogFunc: 节点属性更新.
- sendMsgFunc: 每条边发信息的判断.
- mergeMsgFunc: 融合信息.

类似于 Google Pregel 机制, GraphX 的 Pregel API 运行流程如算法 1 所示.

算法 1 Pregel API 运行流程

```
for all i ∈ nodes do
    vprogFunc(i, initialMsg);
end for
repeat
    for all edge ∈ edges do
        sendMsgFunc(edge);
    end for
    for all i ∈ nodes do
        mergeMsgFunc(Msg a, Msg b);
    end for
    for all i ∈ nodes do
        vprogFunc(value, Msg);
    end for
until No more Msg produced;
```

初始化时, initialMsg 会直接发给每个节点, 节点根据 vprogFunc 更新属性. 之后在每个超步中执行以下操作: 循环每条边, 如需要发信息, 在函数中设定返回值为目标节点 ID 和信息内容. 等信息全部发送完成, 每个收到信息的节点将多个信息融合成一个信息, 不改变信息格式. 最后, 根据信息内容更新节点属性. 之后不断地执行一个个超步, 直到没有任何节点收到新的信息.

节点属性定义: 三元素数组 value[3], 如图 7 所示.

- value(0): 0, 易感; 1, 阅读无传播; 2, 阅读有传播; 3 恢复.
- value(1): 谁传播的 (记录传播源).
- value(2): 第几步 (记录传播时间).

消息定义: 三元素数组 Msg[3], 如图 8 所示.

- Msg(0): 消息类别标志, 0, 初始化; 2, 传播信息; 4, 恢复.
- Msg(1): 这一元素的含义根据消息类别的变化而改变. 如果该消息是传播信息的消息, 那就记录谁传播的 (记录传播源); 如果该信息是初始化信息, 这一元素记录的就是将初始化为传染源的节点编号.
- Msg(2): 记录当前是第几步 (记录传播时间).

对任何一条边都执行发消息操作, 具体规则如算法 2. 节点更新函数如算法 3 所示.

该算法的设计目的和主要特点是各个节点的消息发送和节点更新函数是可以并行处理的, 在有足够的计算资源的情况下可以加速程序运行. 同时在运行过程中不会产生新的文件记录, 所有的行为和

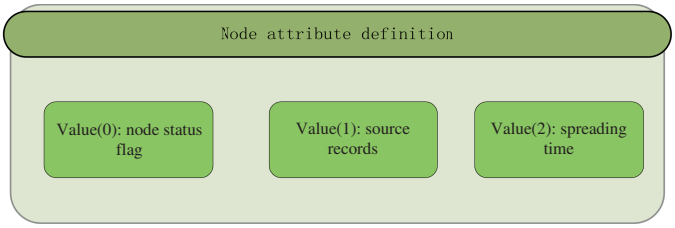

图 7 (网络版彩图) 节点属性定义

Figure 7 (Color online) Node attribute definition

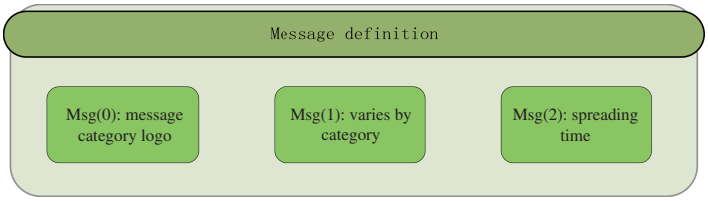

图 8 (网络版彩图) 信息定义

Figure 8 (Color online) Message definition

算法 2 sendMsgFunc

Require: dstId, srcId, dstAttr, srcAttr;
Ensure: Iterator;
1: **if** srcAttr(0) = 0 & dstAttr(0)= 2 **then**
2: **return** Iterator(dstId, Array(4, 0, 0));
3: **else**
4: **if** srcAttr(0) = 2, dstAttr(0)= 0 **then**
5: **return** Iterator(dstId, Array(2, srcId, srcAttr(2)+1));
6: **else**
7: **return** Iterator.empty;
8: **end if**
9: **end if**

状态记录都借助消息传递和节点属性的更新记录在网络中. 最终根据节点属性筛选出被感染的节点, 其属性值就是传播过程的行为记录. 进行实验环境的开发过程中代价比较小. 结果记录和后期的分析也非常简单, 计算过程中不增加额外的文件记录开销, 一定程度上也可以提高计算性能.

4 性能检验

本节首先设计了两种仿真方法的对比实验. 通过对比实验来验证在相同网络中运行同样的模型时该仿真方法相对于使用 Nepidemix 组件进行传播仿真的性能方面的优势.

在进行传播实验之前, 必须清楚在实验中模型和网络是一致的. 生成网络的过程也是仿真运行的重要组成部分. 本文采用度分布符合幂律分布的网络进行实验. 用生成配置网络的方法 (如算法 4) 进行网络生成, 即网络生成分成两个过程, 首先要生成网络节点的度序列, 然后根据该度序列生成网络的边列表. 生成度序列的算法如算法 5 所示. 生成度序列之后需要按照度序列的条件生成边列表, 确

算法 3 vprogFunc

Require: nodeId, Value, Msg, β, γ;
Ensure: Value;
if Msg(0) = 0 **then**
  if nodeId = Value(1) **then**
    return Array(2, 0, 0);
  else
    return Array(0, 0, 0);
  end if
else
  choose a random $p \in (0, 1)$;
  if $p < \beta$ **then**
    choose a random $q \in (0, 1)$;
    if $q < \gamma$ **then**
      return Array(2, Msg(1), Msg(2));
    else
      return Array(3, Msg(1), Msg(2));
    end if
  else
    return Array(0, 0, 0);
  end if
end if

算法 4 配置网络生成

Require: degree[n];
Ensure: EDGE;
index = 0;
for all $i \in (1, n)$ **do**
  for all $j \in (1, \text{degree})$ **do**
    nodelist[index] = i;
    index = index + 1;
  end for
end for
while nodelist is not empty **do**
  choose two random $m, n \in$ (1, size of nodelist), $m \neq n$;
  edge=(nodelist[m], nodelist[n]);
  nodelist[m]=nodelist[last];
  nodelist[n]=nodelist[last−1];
  delete nodelist[last−1];
  delete nodelist[last];
  push edge in EDGE;
end while
return EDGE;

保每个节点在边列表中出现的次数与度相同.

采用 100 核 Spark 集群和单机运行 Nepidemix 进行对比实验. 这 3 组实验主要是验证不同平台下, 节点规模、平均度、感染系数对仿真速度的影响. 统计性能包括网络生成时间和传播计算时间. 实验包括以下内容.

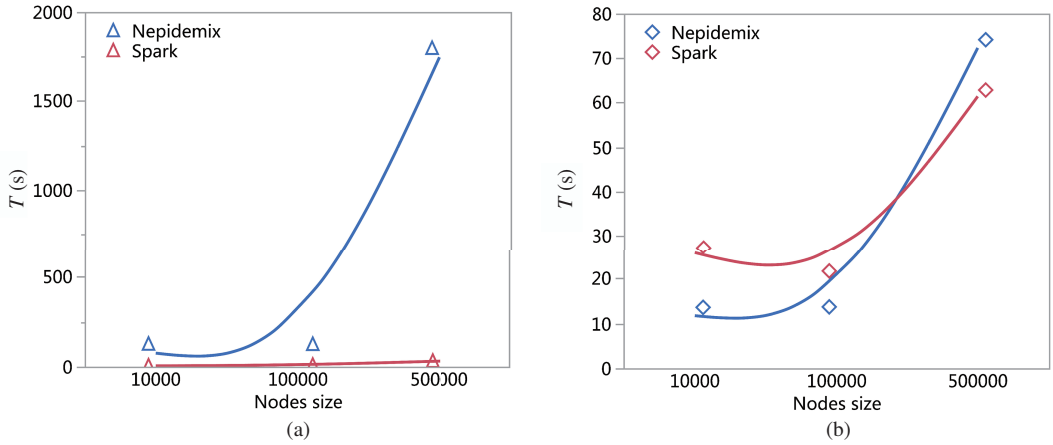

图 9 (网络版彩图) 不同规模对计算时间的影响 (最小度为 6, 感染系数为 0.4). (a) 网络生成时间; (b) 传播计算时间

Figure 9 (Color online) The impact of different nodes size on calculation time (minimum degree 6, infection factor 0.4). (a) Network generating time; (b) propagate calculation time

算法 5 度序列生成

Require: $n \geqslant 0, \text{kmin} > 0, \text{kmax} \geqslant \text{kmin}, \lambda$;
Ensure: degree[n];
1: sum = 0;
2: **for all** $i \in (\text{kmin}, \text{kmax})$ **do**
3: sum = sum + $i^{-\lambda}$;
4: cumpro[i] = sum;
5: **end for**
6: **for all** $j \in (\text{kmin}, \text{kmax})$ **do**
7: cumpro[j] = cumpro[j]/sum;
8: **end for**
9: **for all** $k \in (1, n)$ **do**
10: choose a new random $p \in (0, 1)$;
11: degree[k] = 0;
12: **for all** $l \in (\text{kmin}, \text{kmax})$ **do**
13: **if** cumpro[l] < p **then**
14: degree[k] = degree[k] + 1;
15: **end if**
16: **end for**
17: **end for**
18: **return** degree;

(1) 假定平均度和感染系数不变, 不同节点规模统计、网络生成和传播计算时间, 结果如图 9. 图中可以看出随着节点规模的增大, 使用 Nepidemix 相关组件生成网络需要的时间增加非常快, 但是使用 Spark 平台实验生成网络所需时间受节点规模影响变化非常小, 体现了该平台下网络计算的优势. 而节点规模对传播计算时间影响比较明显, 呈现出节点规模超过 100000 之后迅速上升的规律, 并且 Nepidemix 组件传播计算时间上升更快. 在规模超过 500000 之后 Spark 传播计算所需时间少于 Nepidemix, 在传播计算方面性能优势体现出来.

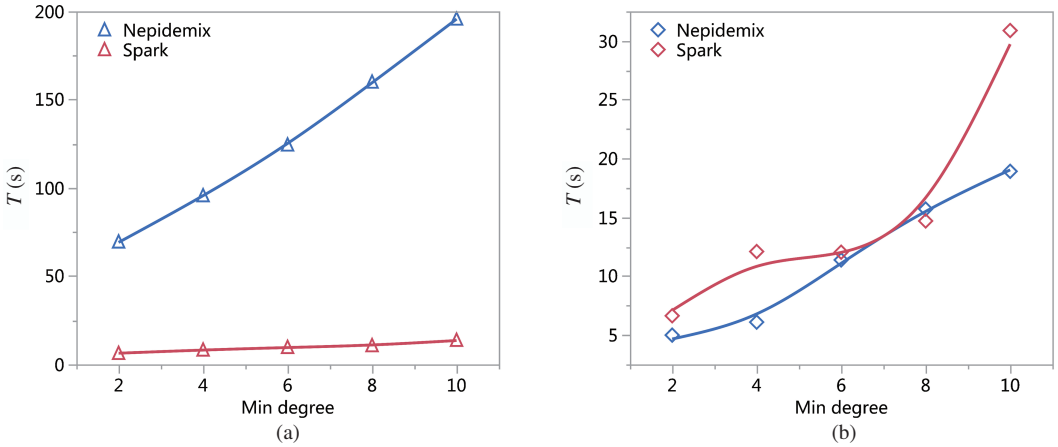

图 10 (网络版彩图) 不同最小度对计算时间的影响 (节点规模为 100000, 感染系数为 0.4). (a) 网络生成时间; (b) 传播计算时间

Figure 10 (Color online) The impact of different minimum degree on calculation time (node size is 100000, infection factor is 0.4). (a) Network generating time; (b) propagate calculation time

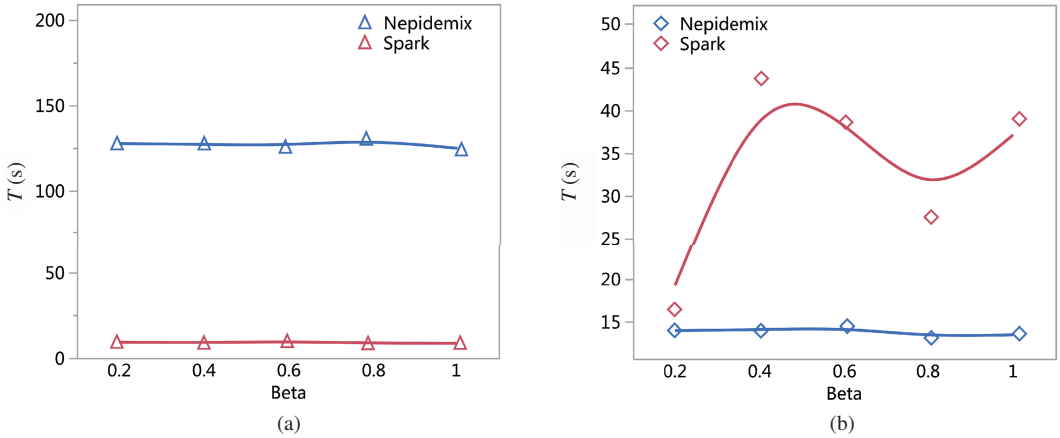

图 11 (网络版彩图) 不同感染系数对计算时间的影响 (节点规模为 100000, 最小度为 6). (a) 网络生成时间; (b) 传播计算时间

Figure 11 (Color online) The impact of different infection factor on calculation time (node size is 100000, minimum degree is 6). (a) Network generating time; (b) propagate calculation time

(2) 假定节点规模和感染系数不变, 不同的最小度下, 网络生成和传播计算时间如图 10(a) 和 10(b). 从图中可以看出, 使用 Nepidemix 相关网络生成时间随着网络最小度的增加而增加得很快, 几乎是线性增加的. 而使用 Spark 平台生成网络需要的时间几乎不随着最小度的变化而变化, 从图中只能看到微弱的上升, 再次体现出网络生成方面的优势. 而关于传播计算时间, 两种实验方法中最小度的影响都比较明显, 但发现 Spark 平台传播计算在度比较大的情况下耗时更多, 这可能是集群上消息传播需要经过网络协议进行信息传递, 并且因为是整体同步并行机制, 整体运行速度完全取决于最慢的集群节点的原因.

(3) 假定最小度和节点规模不变, 不同感染系数下网络生成和传播计算的时间如图 11(a) 和 11(b). 可以看到感染系数对网络生成时间没有影响, 对传播计算的影响也不明显.

整体而言, 采用该平台进行实验并没有在传播计算时间上体现出很明显的优势, 但是却在网络生成时间上体现出了明显的优势, 在多种情况下均发现集群生成网络速度更快. 从实验可以看出 3 个因素对两种方法的网络生成时间和传播计算时间影响有很大的不同. Nepidemix 组件实验网络生成时间受影响变化最明显, 随着节点规模和最小度的增加而明显增加. 同时相对而言 Nepidemix 组件在规模较大时网络生成需要的时间比 Spark 平台多很多. 但在传播计算时间上优势并不明显.

然而, 由于研究条件限制, 本文中对比的还仅仅是百万级别的网络传播计算, 从最大规模的限制来说, 单机 Nepidemix 在千万级别的传播计算需要的时间至少在十天以上, 更大规模情况下基本无法运行, 内存已经溢出. 但是 Spark 平台理论上内存和 CPU 等计算资源可以做到单机的千倍甚至万倍, 在有足够计算资源的情况下, 规模对其影响并不特别突出. 就目前十个节点而言, 测试发现千万级别的网络生成时间在十分钟之内, 单步传播计算时间在一分钟左右, 相对而言可能比小规模情况下慢一些, 但在规模上已经有相当明显的突破, 并且是集群条件限制比较大的情况. 本文的工作很大程度上并没有发挥并行系统的优势, 更多计算资源必然会带来性能的提升, 因此该方法在进行大规模网络传播计算实验方面潜力巨大, 需要较好的条件开展更多工作才能验证.

5 总结

本文介绍了 SIR 模型、SVFR 传播模型以及 Pregel 机制, 以 Pregel 机制为基础并基于 Spark 大数据平台设计并实现了大规模复杂网络传播计算仿真实验算法.

为了探索该算法相对与其他方法的不同, 本文将该算法与 Nepidemix 组件在相同模型和复杂网络条件下进行实验性能对比. 发现该方法生成网络速度有很大优势, 而在传播计算速度上没有明显优势. 但考虑到该方法的性能在有足够计算资源的情况下规模影响并不明显, 还是可以从规模上体现出其性能优势.

本文使用 Spark 平台时限制比较明显, 计算资源有限, 千万级别规模的实验只进行了可行性验证, 没有测试计算性能, 未来在计算资源足够的情况下还应当进行更大规模的实验, 十亿规模网络的传播实验是接下来工作的重点.

Spark 是第一个真正开源并且入门难度较低的大数据平台, 用该平台进行大规模网络相关的计算必然带来很大的优势, 为疾病传播、舆情传播等仿真实验带来了新的方法, 具有明显的优势, 在突破网络计算规模限制方面有非常重要的价值, 为进行大量的多样化的网络传播实验奠定了基础.

大规模图计算平台还有 GraphLab 及其第二代产品 PowerGraph, 其 GAS (Gather, Apply, Scatter) 流程非常适合异步并行计算, 这为开发更贴近真实信息传播情况的新模型奠定了基础. 这些新的模型和技术会为大规模舆情传播仿真带来新的思路和研究方式.

参考文献

1 Watts D J, Strogatz S H. Collective dynamics of 'small-world' networks. Nature, 1998, 393: 440–442
2 Barabási A L, Albert R, Jeong H. Mean field theory for scale-free random networks. Physica A-Statistical Mech its Appl, 1999, 272: 173–187
3 Pastor-Satorras R, Vespignani A. Epidemic spreading in scale-free networks. Phys Rev Lett, 2001, 86: 3200–3203
4 Lloyd A L. Epidemiology: how viruses spread among computers and people. Science, 2001, 292: 1316–1317
5 May R M, Lloyd A L. Infection dynamics on scale-free networks. Phys Rev E, 2001, 64: 066112
6 Granovetter M. Threshold models of collective behavior. Am J Sociology, 1978, 83: 1420–1443

7 Li Q, Braunstein L A, Wang H, et al. Non-consensus opinion models on complex networks. J Stat Phys, 2013, 151: 92–112
8 Qu B, Li Q, Havlin S, et al. Nonconsensus opinion model on directed networks. Phys Rev E, 2014, 90: 052811
9 Goldenberg J, Libai B. Using complex systems analysis to advance marketing theory development: modeling heterogeneity effects on new product growth through stochastic cellular automata. Acad Market Sci Rev, 2001, 9: 1–18
10 Goldenberg J, Libai B, Muller E. Talk of the network: a complex systems look at the underlying process of word-of-mouth. Marketing Lett, 2001, 12: 211–223
11 Hethcote H W. The mathematics of infectious diseases. SIAM Rev, 2000, 42: 599–653
12 Newman M E J. Spread of epidemic disease on networks. Phys Rev E, 2002, 66: 016128
13 Chopard B, Droz M. Cellular Automata Modeling of Physical Systems. Cambridge: Cambridge University Press, 1998
14 Zhang M X, Seck M, Verbraeck A. A DEVS-based M&S method for large-scale multi-agent systems. In: Proceedings of the 2013 Summer Computer Simulation Conference, Toronto, 2013. 3
15 Collier N, Ozik J, Macal C M. Large-scale agent-based modeling with repast HPC: a case study in parallelizing an agent-based model. In: Euro-Par 2015: Parallel Processing Workshops. Berlin: Springer, 2015. 454–465
16 Seck M, Verbraeck A. Devs in dsol: adding devs operational semantics to a generic event-scheduling simulation environment. In: Proceedings of the Summer Computer Simulation Conference, Istanbul, 2009. 261–266
17 Staudt C L, Sazonovs A, Meyerhenke H. Networkit: an interactive tool suite for high-performance network analysis. ArXiv: 1403.3005, 2015
18 Bastian M, Heymann S, Jacomy M. Gephi: an open source software for exploring and manipulating networks. In: Proceedings of the 3rd International AAAI Conference On Weblogs And Social Media, San Jose, 2009
19 Batagelj V, Mrvar A. Pajek—analysis and visualization of large networks. In: Graph Drawing Software. Berlin: Springer, 2004. 2265: 77–103
20 Leskovec J, Sosic R. Snap: a general-purpose network analysis and graph-mining library. ACM Trans Intell Syst Tech, 2016, 8: 1
21 Lugowski A, Alber D, Buluç A, et al. A flexible open-source toolbox for scalable complex graph analysis. In: Proceedings of the 2012 SIAM International Conference on Data Mining, Anaheim, 2012
22 Ediger D, Jiang K, Riedy E J, et al. Graphct: multithreaded algorithms for massive graph analysis. IEEE Trans Parall Distrib Syst, 2013, 24: 2220–2229
23 Ediger D, Mccoll R, Riedy J, et al. Stinger: high performance data structure for streaming graphs. In: Proceedings of 2012 IEEE Conference on High Performance Extreme Computing, Waltham, 2013. 1–5
24 Shun J L, Blelloch G E. Ligra: a lightweight graph processing framework for shared memory. In: Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), Shenzhen, 2013. 135–146
25 Rutherford A R, Ramadanović B, Ahrenberg L, et al. Control of an HIV epidemic among injection drug users: simulation modeling on complex networks. In: Proceedings of Winter Simulation Conference, Arlington, 2016. 23–37
26 Capasso V, Serio G. A generalization of the Kermack-McKendrick deterministic epidemic model. Math Biosci, 1978, 42: 43–61
27 Liu L, Qu B, Chen B, et al. Modeling of information diffusion on social networks with applications to wechat. ArXiv: 1602.00193, 2017
28 Goudreau M W, Lang K, Rao S B, et al. Portable and efficient parallel computing using the BSP model. IEEE Trans Comput, 1999, 48: 670–689
29 Hill J M D, McColl B, Stefanescu D C, et al. BSPlib: the BSP programming library. Parall Comput, 1998, 24: 1947–1980
30 Skillicorn D B, Hill J M D, Mccoll W F. Questions and answers about BSP. Sci Programm, 1997, 6: 249–274
31 Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing. In: Proceedings of ACM SIGMOD International Conference on Management of Data, Indianapolis, 2010. 135–146
32 Gonzalez J E, Low Y C, Gu H J, et al. Powergraph: distributed graph-parallel computation on natural graphs. In: Proceedings of Usenix Conference on Operating Systems Design and Implementation, Hollywood, 2012. 17–30
33 Gonzalez J E, Xin R S, Dave A, et al. GraphX: graph processing in a distributed dataflow framework. In: Proceedings of Usenix Conference on Operating Systems Design and Implementation, Broomfield, 2014. 599–613
34 Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets. In: Proceedings of the

2nd Usenix Conference on Hot Topics in Cloud Computing, Berkeley, 2010. 10–10
35 Valiant L G. A bridging model for parallel computation. Commun ACM, 1990, 33: 103–111

Design and implementation of large-scale network propagation simulation method inspired by Pregel mechanism

Chuan AI[1], Bin CHEN[1*], Liang LIU[1], Jian DONG[1], Lingnan HE[2], Kaisheng LAI[2] & Xiaogang QIU[1]

1. *College of System Engineering, National University of Defense Technology, Changsha* 410073*, China*;
2. *School of Communication and Design, Sun Yat-sen University, Guangzhou* 510006*, China*
* Corresponding author. E-mail: nudtcb9372@gmail.com

Abstract With the rapid development of the Internet and online social media, the law of information dissemination in social networks needs much experimentation on network propagation calculation. The current network propagation experiments based on the SIR model are widely used in disease research and information dissemination. However, because of the limitations of hardware and software, it is still difficult to conduct ultra-large-scale network propagation calculation. However, the current Internet information dissemination shows the characteristics of large-scale users, large amounts of information, and fast propagation. The shortcomings of small-scale network dissemination experiments based on abstract and simplified methods have been revealed. In this study, the Spark platform is used to implement an experimental algorithm for large-scale network propagation calculation. The performances of the algorithm and Nepidemix stand-alone computing components are compared, and the algorithm's advantages and disadvantages are demonstrated. An orthogonal experimental design method was used to design the performance test experiment to find out the factors influencing the algorithm. When there are enough cluster computing resources, the algorithm can break the limitation of network node size and is difficult to develop, which lays the foundation for calculation experiments about very large-scale network propagation.

Keywords information dissemination, distributed computing, SIR, complex networks, simulation method

Chuan AI obtained his master's degree from the National University of Defense Technology, China in 2016. He is studying for a Ph.D. degree at the National University of Defense Technology. His main research direction includes the modeling and simulation of the propagation of unconventional online public opinion.

Bin CHEN obtained his Ph.D. degree from the National University of Defense Technology, China in 2010. He is currently a research associate at the College of Information Systems and Management, National University of Defense Technology. His research focuses on emergency management of unconventional emergencies, modeling and simulation of network public opinion, and multiagent modeling and simulation.