基于社团并行发现的在线社交网络蠕虫抑制

和 亮 冯登国 苏璞睿 应凌云 杨 轶

(中国科学院软件研究所可信计算与信息保障实验室 北京 100190)

摘 要 随着在线社交网络(Online Social Network,OSN)的快速发展,OSN 蠕虫已经成为最具威胁的网络安全问题之一.为了防止 OSN 蠕虫的快速传播,文中提出了一种基于社团并行发现的 OSN 蠕虫抑制方法.首先将分布式图计算框架 Pregel 和基于标签传播的社团发现算法(Label Propagation Algorithm,LPA)相结合,提出了一种能够处理大规模 OSN 网络社团发现问题的并行 LPA 算法(Parallel LPA,PLPA).其次,文中在 PLPA 算法的基础上给出了 3 种社团关键节点的选取策略,并提出了相应的 OSN 蠕虫抑制方法.最后,通过在两组真实数据集上进行的社团并行发现及 OSN 蠕虫抑制仿真实验证明了文中方法的有效性.

关键词 社团并行发现;在线社交网络;蠕虫抑制;社会计算;社交网络

中图法分类号 TP309 **DOI 号** 10.3724/SP.J.1016.2015.00846

Parallel Community Detection Based Worm Containment in Online Social Network

HE Liang FENG Deng-Guo SU Pu-Rui YING Ling-Yun YANG Yi

(*Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

Abstract With the rapid development of Online Social Networks (OSNs), worms propagating in these networks have become one of the most threatening security problems. To contain these rapidly spreading worms, in this paper, we propose a defensive measure which is based on parallel community detection in OSNs. Specifically, according to the Pregel data-processing infrastructure, we implement a new parallel version of label propagation algorithm (PLPA) that is capable of quickly finding communities in OSNs owning millions of users. And then we give three definitions for the influential users to whom we will first distribute patches to contain the propagation of OSN worms. To evaluate the performance of our approaches we test them on our simulating framework with two large-scale OSN datasets and analyze the experimental results which can show the effectiveness of our approaches.

Keywords parallel community detection; online social network; worm containment; social computing; social network

1 引 言

目前,在线社交网络(Online Social Network,OSN)已经成为互联网上最为流行的服务之一.但与此同时,随着 OSN 的大规模流行,一些新的安全问题也开始不断涌现.其中,新型的 OSN 蠕虫以其独特的传播方式成为目前最具威胁的安全问题之一.

收稿日期:2013-07-06;最终修改稿收到日期:2014-11-02.本课题得到国家"九七三"重点基础研究发展规划项目基金(2012CB315804)、国家自然科学基金(61073179)、国家自然科学基金重大研究计划(91118006)、北京市自然科学基金(4122086).**和 亮**,男,1985 年生,助理研究员,主要研究方向为信息安全、社交网络恶意代码分析.E-mail:heliang@tca.iscas.ac.cn.**冯登国**,男,1965 年生,博士,研究员,主要研究领域为密码学、信息安全.**苏璞睿**,男,1976 年生,博士,研究员,主要研究领域为网络与系统安全.**应凌云**,男,1982 年生,博士,高级工程师,主要研究方向为僵尸网络分析.**杨 轶**,男,1983 年生,博士,助理研究员,主要研究方向为网络与系统安全.

与传统蠕虫不同，OSN 蠕虫主要是通过社会工程以各种方式欺骗用户点击而进行快速传播. 例如，2011 年 6 月 HelloSamy 蠕虫通过利用新浪微博中存在的跨站脚本漏洞，在 15 分钟内感染 3 万多用户①. 其传播过程主要是，依靠在被感染用户的页面上自动发布带有恶意短链接且具有很强欺骗性的消息，当其他好友因好奇而浏览该消息并点击其中的短链接时，就会使得本地浏览器自动加载并执行相关恶意代码，进而使得该好友被蠕虫感染. 在 2013 年 4 月份爆发的 Twitter Ohaa 蠕虫也采取了同样类似的传播方式，在短时间内感染了大量用户②. 此外，OSN 用户之间较强的信任关系以及 OSN 网络拓扑本身所具有的小世界[1]和无标度[2]等特性同样在很大程度上加快了 OSN 蠕虫的传播，具体原因可参见 6.2 节的相关描述.

针对目前 OSN 蠕虫所造成各种危害，有研究人员提出了初步的防御措施. 具体来说，主要包括两个方面：OSN 蠕虫检测和 OSN 蠕虫抑制. 其中，OSN 蠕虫检测可根据检测的位置分为服务端检测[3-5]和客户端检测[6]. 前者主要是通过在 OSN 网站的服务器端捕获网络中被判定为恶意的消息的数量，但该方法只有当恶意消息扩散到一定程度后才能检测到 OSN 蠕虫. 由此可见，该方法具有不可避免的延迟性. 而客户端检测方法主要是利用不断更新的特征库来匹配并检测出 OSN 蠕虫传播的情况. 但当有新类型蠕虫爆发时，受限于现有网络带宽等条件很难及时将新的特征内容同时下发到网络中所有用户的检测系统中，故该方案也存在不可避免的延迟性.

考虑到检测方法所具有的延迟性，目前已有研究人员提出将 OSN 蠕虫抑制方法[7-9]作为另一种有效的防御措施. 虽然无法及时检测出 OSN 蠕虫的传播，但是应该提出一种能够最大程度上减少受感染用户数量的解决方法. 而目前研究人员主要普遍采用的方案是从复杂网络自身具有的社团结构特性[10]来入手解决，即首先寻找网络中的社团，之后根据相关度量手段（例如节点的度数）来选取社团中的关键节点. 最后，通过优先对这些关键节点来进行免疫操作，从而保证可以以最快速度来保证其他节点得到免疫.

目前，寻找复杂网络中的社团结构已经成为当前相关领域的研究热点之一，并且也有大量的社团发现算法被提出. 这些方法可分为非重叠社团发现方法和重叠社团发现方法. 其中非重叠社团发现方法中较为典型的有：基于分裂思想的 GN 算法[10]、基于凝聚思想的 FN 算法[11]以及基于标签传播的 LPA 算法[12]；而在重叠社团发现方法中，由 Palla 等人[13]首次提出了基于完全子图的 CPM 算法. 之后又有大量的研究人员在此基础上给出了各自具有代表性的工作，例如，基于凝聚思想的 EAGLE 算法[14]、基于贪心思想的 GCE 算法[15]以及基于扩展标签传播思想的 COPRA 算法[16].

由于目前主流的 OSN 网站（Facebook、Twitter、新浪微博、朋友圈等）中的用户数量已经达到上亿（甚至十亿）规模③，而通过对以上各类算法的研究发现，为了使算法能够处理如此大规模的网络，研究人员只是考虑如何优化自身的算法. 例如，作为目前最为流行的非重叠社团发现算法之一，BGLL 算法[17]通过计算局部节点信息来提高其运行效率. 但由于该算法只能在单个计算节点中运行，即不具备可扩展性，因此单个节点的计算能力决定了该算法的实际处理能力.

另一方面，为了使得现有算法具备可扩展性，研究人员利用了分布式技术来解决大规模网络中社团发现问题，例如 Chen 等人[18]和 Yang 等人[19]分别使用 MapReduce[20]技术重新实现了 FN 算法以及 CPM 算法. 但通过分析其具体实现过程可以发现，MapReduce 的“易并行”特性并不适合处理复杂的基于图的计算问题，其主要原因是该技术需要在前一个处理过程中保存整个图的状态并作为下一个任务的输入，而这样处理则会降低整个计算任务的运行效率.

本文将 Google 提出的分布式图计算框架 Pregel[21-22]运用到非重叠社团发现的 LPA 算法中，通过解决 MapReduce 难以处理图并行计算的问题，提出了一种适合处理大规模 OSN 的社团并行发现算法 PLPA（Parallel Label Propagation Algorithm），并将其应用到 OSN 蠕虫抑制的过程中.

本文的主要贡献总结如下：

（1）提出了一种基于 Pregel 模型的社团并行发现方法. 该方法主要利用了分布式技术实现了可并行化的 PLPA 算法，使其能够有效处理大规模网络中的社团发现问题.

（2）提出了一种基于社团并行发现的 OSN 蠕

① http://baike.baidu.cn/view/6001028.htm
② http://twitter.com/kadiraltan
③ http://en.wikipedia.org/wiki/List_of_social_networking_websites

虫抑制方法.该方法主要在 PLPA 算法的基础上,通过不同策略来选取社团中的关键节点进行优先免疫,从而达到对 OSN 蠕虫的及时抑制.

(3)通过仿真实验验证了本文方法的有效性.本文结合真实的大规模数据集,对本文方法的有效性进行了实验检验.结果表明,我们所提出的 PLPA 算法能够快速有效的发现社团结构,同时相应的 OSN 蠕虫抑制方法能够取得较好的防御效果.

本文第 2 节介绍 OSN 蠕虫抑制和社团发现的相关工作;第 3 节阐述本文方法的总体框架;第 4 节和第 5 节分别给出 PLPA 算法的具体实现及相应的 OSN 蠕虫抑制方法;第 6 节介绍在真实数据集上的实验效果;第 7 节总结全文工作.

2 相关工作

OSN 蠕虫抑制相关的工作最初是由 Zhu 等人[7]在研究移动电话网络中如何快速发布补丁所提出来的.其主要是考虑到在蠕虫爆发时无法在短时间内将补丁下发给所有移动用户,因此需要提出一种分步的快速下发机制.而其核心思想是借助移动网络中的用户关系,通过提取社交关系图并对其进行划分,之后在所划分的图上利用其提出的最少分割节点算法来进行补丁下发,从而达到及时有效的蠕虫抑制效果.

而基于社团结构的 OSN 蠕虫抑制方法则是由 Nguyen 等人[8]首先提出来的.其主要考虑到基于图划分的方法很难反映真实的网络结构特性,同时该方法在划分过程中必须要由用户提供划分的数量 k,而事实上该参数理应是由划分算法给出.因此,Nguyen 等人主要的贡献是一方面提出了基于社团发现的 OSN 蠕虫抑制方法,该方法利用目前流行的 BGLL 社团发现算法能够在不需要用户提供划分数量 k 的情况下即可找出合理的社团结构.另一方面作者给出了社团中关键节点的选取策略,即将社团中和其他社团连接数最多的节点定义为关键节点,并优先对这些节点进行免疫或者补丁下发.但在文章中,作者并没有对关键节点的选取进行理论证明或者实验说明.

此外,Nguyen 等人[9]还提出了一种基于局部密度函数的重叠社团发现方法.与以往算法不同的是,作者在发现社团的过程中考虑了用户关系的动态调整的问题.最后,作者在此基础上给出了相应的 OSN 蠕虫抑制策略,即在找到重叠社团以后,通过对重叠区的邻居节点进行补丁分发,从而能够达到更为有效的 OSN 蠕虫抑制效果.

可以看出,为了能够较好的抑制 OSN 蠕虫传播,高效寻找网络中的社团结构是十分关键的.而为了能够有效处理大规模网络数据,目前已经有学者提出使用云计算中的分布式并行计算技术来解决相应的大规模问题.

Chen 等人[18]首次提出利用 MapReduce 技术来重新实现基于凝聚思想的 FN 算法.其主要思路是利用 MapReduce 求出任意节点对之间的模块度增益,之后找出增益最大的节点对并将其合并.这样处理的效率仍然很低,因为其算法每次迭代只能处理一对节点.虽然作者随后提出了一次处理多对节点的优化方法,但这样的处理方式会对结果的准确性造成很大影响.此外,Yang 等人[19]提出通过利用 MapReduce 来发现极大完全子图的方式重新实现了 CPM 算法.其主要思路是先将原始图预处理为邻接表的形式,之后在邻接表上构建完全子图,并通过这些完全子图来发现重叠的社团结构.

Zhang 等人[23]首次提出了利用整体同步模型(Bulk Synchronous Parallel,BSP)[24]来解决无向无权图中的重叠社团发现问题.其主要思路是首先给出节点之间邻接(propinquity)的概念,进而结合拓扑自身的结构特性给出了基于邻接动态特性的重叠社团发现算法.该算法的缺陷是需要用户定义不同的参数来确定节点之间是否为邻接,这也意味着社团发现结果的准确性很大程度上会受到用户参数的影响.之后,Zhang 等人[25]提出了利用 BSP 计算模型来解决有向带权图中的重叠社团发现问题.其核心思路是依据文献[26]给出的概念,首先找出自然社团,之后通过迭代来合并两两相似的自然社团,从而发现有向带权图中的重叠社团.该方法仍然存在效率问题,因为每次迭代只能合并两个相似的社团,而每一次迭代都需要很大的计算开销.

综上所述,虽然 Nguyen 等人提出的基于社团发现的 OSN 蠕虫抑制方案能够在一定规模的网络中起到较好的防御作用,但是仍然存在两个方面不足:首先,由于作者所使用社团发现算法本身不具备可扩展性,因此无法有效应对大规模网络.其次,作者在文中只给出了单一的关键节点选取策略,而没有给出其他选取策略及相关抑制效果的比较.此外,虽然已有部分可扩展社团发现方法被提出,但通过

以上分析可知，这些算法仍然存在运行效率较低的问题.

3　总体框架

本文在现有方法的基础上，提出了一种基于并行化社团挖掘的 OSN 蠕虫抑制方法. 如图 1 所示，该方法主要分为 4 个阶段：原始数据预处理、并行社团发现、关键节点提取以及 OSN 蠕虫快速抑制.

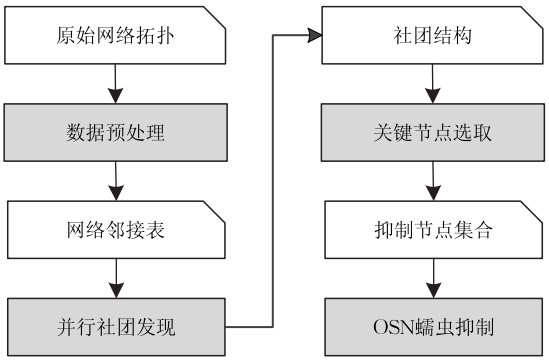


图 1　OSN 蠕虫抑制主要流程

(1) 原始数据预处理. 由于从实际网络中提取到的原始拓扑结构的存储格式多种多样，并且一般规模较大，因此需要预处理来进行数据格式的统一. 本文利用 MapReduce 并行化技术将原始数据统一处理成邻接表格式的网络拓扑结构. 具体来说，我们假设原始输入是基于边集合的存储格式，用 $\{\langle n1,n2\rangle\}$ 来表示，其中 $n1$ 和 $n2$ 表示网络中的任意两个节点. 本文在数据预处理阶段主要实现的过程可以用如下形式来表示：

Mapper：$\langle n1,n2\rangle \rightarrow \langle k,v\rangle$

Reducer：$\langle k,\text{list}(v)\rangle \rightarrow \langle k,v_1-v_2-\cdots-v_n\rangle$

在上述表达式中的 Mapper 方法将原始网络中的每一条边转换为可供处理的 key-value 二元组(这里简记为 k 和 v). 而 Reducer 方法则负责收集每一个节点 k 对应的邻居节点，进而能够得到该节点的邻接表，并以 $v_1-v_2-\cdots-v_n$ 形式表示. 关于 MapReduce 的具体处理流程可参见文献[20]中的具体描述.

(2) 并行社团发现. 该阶段是 OSN 蠕虫抑制处理过程中的核心过程，其能否快速准确提取出大规模网络中的社团结构，决定了最终 OSN 蠕虫抑制的效果. 本文通过利用 Pregel 计算模型，将预处理阶段得到的网络邻接表作为输入，通过并行化计算，将网络中的社团结构作为输出，具体算法描述将在第 4 节中详细介绍.

(3) 关键节点提取. 当并行化社团计算完成后，本文将对社团结构中具有重要属性特征的节点(称之为“社团关键节点”). 而根据不同属性特征的选取，本文提出了 3 种不同的关键节点选取策略，即基于最大度数节点、最大内部度数节点以及最大外部度数节点. 不同的关键节点会导致不同的抑制效果，具体选取策略将会在第 5 节介绍.

(4) OSN 蠕虫抑制. 当利用关键节点选取策略筛选出所有社团结构中的关键节点后，本文将对其进行优先免疫操作，之后再借助这些关键节点来对其周围邻居节点进行免疫，进而保证能够有效抑制 OSN 蠕虫的快速传播. 为了说明选取不同抑制策略所导致的 OSN 蠕虫抑制效果，本文将会在第 6 节详细描述相关对比实验结果.

4　基于 Pregel 的社团并行发现

本节详细介绍基于图分布式计算技术——Pregel 的社团并行发现算法. 首先，介绍 Pregel 模型及利用该模型处理问题的一般流程. 其次，在分析基于标签传播的社团发现算法可扩展性的基础上，提出 PLPA 算法.

4.1　Pregel 模型

Google 研究人员在利用 MapReduce 解决大规模并行处理问题时，发现其很难应用于复杂的图计算领域，因此提出了一种新的基于整体同步并行 BSP 模型，称之为 Pregel 模型[21-22]. 在该模型中，所有待处理的对象均可以用有向图 $G=\langle V,E\rangle$ 来表示，其中顶点集合 $V=\{v_i \mid i=1,2,\cdots,n\}$，有向边集合 $E=\{e_{ij} \mid e_{ij}=\langle v_i,v_j\rangle, v_i,v_j\in V, i,j=1,2,\cdots,n\}$. 而在具体的处理过程中，Pregel 作为一种消息驱动的面向节点的计算模型. 如图 2 所示，虚线即表示节点接收或者发送消息的过程. 另外，在该模型中，待处理图 G 中的所有顶点均被赋予独立计算的能力，

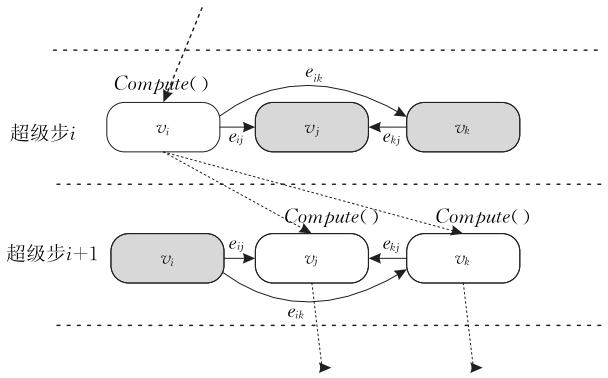


图 2　Pregel 模型中的计算流程

如图 2 所示，顶点上方的 Compute 方法表示该节点的计算过程. 关于 Pregel 的具体处理流程可参见文献[21]中的具体描述.

由于 Pregel 本身是基于 BSP 模型的，因此正如上图所示，整个计算过程可以看作是迭代执行一系列步骤(被称为超级步)的过程. 而在每一次迭代过程中，系统会为每一个接收到消息的活跃顶点(图中白色顶点)执行一次 Compute 方法，该方法则需要由用户根据不同的计算任务来为所有顶点提供统一的实现过程. 本文这里以伪代码的形式总结出 Pregel 模型中所有节点计算时的一般流程，而该流程也作为下文 PLPA 算法的主要框架.

过程 1. Compute 方法.

输入：消息集合 M，当前状态值 S，停机条件 C

输出：是否停机

```
1.  N←∅, R←∅, T←∅
2.  for m_i in M
3.      r_i←calculate(m_i)                    //〈1〉
4.      R.add(r_i)
5.  setStatus(R, S)                           //〈2〉
6.  T←getTargets()                            //〈3〉
7.  M'←createMessages(R, targets)             //创建新消息
8.  for m'_i in M'
9.      send(m_i)
10. if(checkStatus(C))                        //〈4〉
11.     voteToHalt()
```

通过观察以上方法可以看出，节点在执行计算时都需要完成以下 4 项任务，即

〈1〉核心计算(第 2～4 行). 该任务是遍历该顶点在运行时所有接收到的消息内容并进行综合的计算过程，同时该过程也是利用 Pregel 模型来解决各类计算问题的主要处理过程.

〈2〉设置顶点状态(第 5 行). 该任务主要是根据上一个任务计算得到的结果来修改当前的状态值. 用户可以根据不同的任务来定义不同的状态含义. 例如，本文将标签值作为顶点的状态值.

〈3〉创建并发送消息(第 6～9 行). 该任务是通过核心计算任务的结果来创建新的待发送消息集合. 需要说明的是，消息的创建以及消息的发送目标都由用户根据不同的任务、条件来完成，而消息的发送过程则通过系统提供的内部方法来完成.

〈4〉判断停机(第 10～11 行). 该任务是根据当前的状态值来判断是否满足用户提供的停机条件. 若满足，则该节点由活跃状态变为停机状态(图 2 中的灰色顶点)，同时该节点将不会参与下一个超级步中的计算直到有其他节点向其发送消息为止. 而当所有顶点均为停机状态时，整个迭代流程将会终止.

4.2 PLPA 算法

原始 LPA 算法是由 Raghavan 等人为了有效处理较大规模网络而提出来接近线性计算时间的社团发现算法. 该算法与 Pregel 模型整体计算过程相似之处在于均采用迭代执行的过程，即在算法的每一次迭代过程中，所有节点首先收集其周围邻居节点的标签，之后将其自身的标签赋值为出现次数最多的标签值. 整个算法的迭代过程直到所有节点的标签值是其周围邻居节点中个数最多的标签值为止.

在原始 LPA 算法中[12]，作者为了解决自身固有的“振摆”问题①，提出了异步解决策略，在描述该策略之前，下面先给出相关符号的说明：

$L_t(v_i)$：节点 v_i 的 t 时刻标签值；

N_i：表示节点 v_i 的邻居节点集合，并用 n_i 表示该集合的节点数；

f：负责确定某节点标签值的函数，一般将其定义为选取某个标签集合(允许重复)中出现次数最多的标签值；

q：表示异步比例，也称异步程度，即在 t 时刻用 f 来计算节点 v_i 的标签值时，将选取其邻居节点中 $q\times n_i$ 个节点的 $t-1$ 时刻标签值和 $(1-q)\times n_i$ 个节点的 t 时刻的标签值来作为输入参数；

根据以上说明，假设已知 t 时刻顶点 v_i 的标签值 $L_t(v_i)$ 由函数 f 来确定，并且此时 v_i 的邻居节点为 $N_i=\{v_1,v_2,\cdots,v_{n_i}\}$，则 $L_t(v_i)$ 的计算方式如下：

$$L_t(v_i)=f(L_{t-1}(v_1),\cdots,L_{t-1}(v_{q\times n_i}),L_t(v_{q\times n_i+1}),\cdots,L_t(v_{n_i})) \quad (1)$$

在式(1)中，由于输入格式限制，这里 $q\times n_i$ 表示将向下(或上)取整. 此外，根据上式可知，当 $q=0$ 时，意味着在选取标签值时，将采用同步策略，即 t 时刻任一节点的标签值由其周围邻居节点的 t 时刻标签值来确定；当 $0<q<1$ 时，表示算法选择异步策略；当 $q=1$ 时，称为完全异步策略. 本文将在第 6 节实验部分(见图 7(a))分析不同异步程度对算法结果的影响.

而根据以上对 LPA 算法的简要描述可以发现，其不仅与 Pregel 模型的整体迭代过程相似，同时每一次迭代中所有节点都需要执行的计算过程均可以划分为前述的 4 项任务，正是在此基础上，本文提出

① 造成“振摆”问题的原因是在于当图中存在二部图或者近似二部图时，每个节点由于难以拥有稳定的标签值，从而造成“振摆”的出现.

了基于 Pregel 的 PLPA 算法，如下所示.

算法 1.　PLPA 算法.

输入：含有标签值的消息集合 M，节点 ID，异步程度 q

输出：是否停机

```
1.  step←getSuperstepCount() //当前超级步数
2.  if(step %2=0) //一次 LPA 迭代需要两个超级步
3.    if(step=0)   //初始化标签值为节点 ID
4.      currentLabel←ID
5.      lastLabel←ID
6.    else          //计算标签值
7.      maxLabel←getMaxLabel(M)          //⟨1⟩
8.      if(currentLabel!=MaxLabel)       //⟨2⟩
9.        lastLabel←currentLabel
10.       currentLabel←maxLabel
11.     else                             //⟨3⟩
12.       voteToHalt()
13. else
14.   p←getRandomProbability()           //⟨4⟩
15    T←getAllNeighbors() //所有邻居节点
16.   if(p>q)                //判断异步程度
17.     M′←createMessages(currentLabel, T)
18.   else
19.     M′←createMessages(lastLabel, T)
20.   for m_i′ in M′
21.     send(m_i′)
```

这里对 PLPA 算法进行如下说明：

首先，利用 Pregel 模型中的两个超级步来完成原始 LPA 算法中的一次迭代过程. 其中第 1 个超级步完成核心计算任务，即获取周围邻居中个数最多的标签值，见算法第 3～12 行. 而第 2 个超级步需要完成异步标签策略，即通过比较随机概率值 $p(0<p<1)$ 与异步程度 q 来决定节点发送的标签值，这样就可以保证在下一个超级步中其相邻节点可以按照式(1)来计算自身的标签值，见算法 14～19 行.

其次，虽然给出的 PLPA 算法遵循了原始 LPA 算法给出的终止条件(见算法第 8～9 行)，但根据文献[12]中的相关描述及实验证明，算法只需迭代 5 次以上即可保证至少 95%的节点被标记为正确的标签值. 因此在实际的应用中，我们可以将迭代次数是否超过设定阈值作为算法结束的条件，而本文将会在第 6 节实验部分分析不同的迭代次数对算法所产生的影响.

5　OSN 蠕虫抑制方法

本节介绍在 PLPA 算法基础上的 OSN 蠕虫抑制方法. 首先，依据不同的度量方法，提出了 3 种社团关键节点的选取策略；其次，给出了基于社团并行发现的 OSN 蠕虫抑制算法及相关说明.

5.1　关键节点选取策略

由于在基于社团结构的 OSN 蠕虫抑制方法中，除了社团发现算法的效率以外，不同的关键节点选取策略同样会影响 OSN 蠕虫的抑制效果. 下面首先给出本文以节点度数为度量值的关键节点的形式化定义.

假定社交网络 G 由 n 个非重叠社团 C_i 组成，即 $G=\bigcup_{i=1}^{n} C_i$. 此外，用 $|V_j^i|^{\mathrm{In}}$ 表示社团 C_i 中的节点 V_j^i 和本社团内其他节点的连接数，亦称内部节点度数；用 $|V_j^i|^{\mathrm{Out}}$ 表示社团 C_i 中的节点 V_j^i 和其他社团中节点连接的度数，亦称外部节点度数；用 $|V_j^i|$ 表示社团 C_i 中的节点 V_j^i 和其他节点连接的度数，其值等于该节点外部度数和内部度数之和.

定义 1. 最大度数节点. 在社团 C_i 中，称 V_{Max}^i 是其最大度数节点，当且仅当满足如下条件：

$$\forall V_j^i \in C_i,\ |V_j^i| \leq |V_{\mathrm{Max}}^i|.$$

定义 2.　最大内部度数节点. 在社团 C_i 中，称 V_{MaxIn}^i 是其最大内部度数节点，当且仅当其满足如下条件：

$$\forall V_j^i \in C_i,\ |V_j^i|^{\mathrm{In}} \leq |V_{\mathrm{Max}}^i|^{\mathrm{In}}.$$

定义 3.　最大外部度数节点. 在社团 C_i 中，称 V_{MaxOut}^i 是其最大外部度数节点，当且仅当其满足如下条件：

$$\forall V_j^i \in C_i,\ |V_j^i|^{\mathrm{Out}} \leq |V_{\mathrm{Max}}^i|^{\mathrm{Out}}.$$

基于以上 3 个定义，本文给出了如下 3 种不同的关键节点选取策略及相关说明：

(1) Max 策略. 在 OSN 蠕虫爆发时，优先选取每一个社团中最大度数节点 V_{Max}. 该类节点是拥有最多邻居的节点，而邻居节点既可以是同一社团节点也可以是不同社团中的节点. 该选取策略主要是基于贪心思想，即优先免疫局部传播能力最强的节点.

(2) MaxIn 策略. 在 OSN 蠕虫爆发时，优先选取社团中最大内部度数节点 V_{MaxIn}. 该类节点是在社团中拥有和本社团内部其他节点链接最多的节点，选取该类节点是基于局部的考虑，因为该类节点免疫后其可以以最快的速度对本社团内其他节点进行免疫，从而抑制 OSN 蠕虫在本社团内的传播.

(3) MaxOut 策略. 在 OSN 蠕虫爆发时，优先选取社团中最大外部度数节点 V_{MaxOut}. 该类节点是

在社团中拥有和外部其他社团连接最多的节点，选取该类节点主要是从如下两个方面考虑：首先，其可以有效防止外部社团中的 OSN 蠕虫向该社团传播；其次，该类节点可以最大程度抑制社团内部的 OSN 蠕虫向外扩散.

假设在图 3 所示的结构中，虚线框表示社团结构 S，则节点 A 即为 V_{Max} 节点，节点 B 为 V_{MaxIn} 节点，而节点 C 则是 V_{MaxOut} 节点. 尽管在多数情况中，V_{Max} 节点和 V_{MaxIn} 节点以及 V_{MaxOut} 节点有可能是同一个节点，但在图 3 所示的社团结构中，V_{Max} 节点并不是 V_{MaxOut} 节点，同时也不是 V_{MaxIn}. 以上情况正是本文提出 3 种不同选取策略的主要原因.

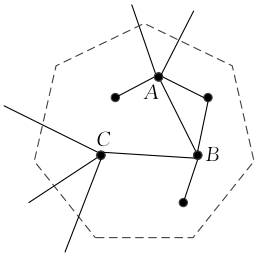


图 3 3 种策略下的关键节点

5.2 OSN 蠕虫抑制算法

当通过 PLPA 算法获取到当前 OSN 的社团结构后，根据上一小节给出的 3 类选取策略，我们给出本文的 OSN 蠕虫抑制算法.

算法 2. OSN 蠕虫抑制算法.

输入：关系集合 E，社团结构 $C=\{C_i\}$，选取策略 P

输出：关键节点集合 S

```
1.  S←∅
2.  for C_i in C
3.      if (P 为 Max 策略)
4.          v←get_max(C_i, E)
5.      elseif (P 为 MaxIn 策略)
6.          v←get_max_in(C_i, E)
7.      else
8.          v←get_max_out(C_i, E)
9.      if (v!=NULL)
10.         S.add(v)
11. for v in S
12.     向 v 下发免疫通告或者修复补丁
13.     v 免疫后向其邻居扩散通告或补丁
14. return S
```

这里对于算法 2 进行如下补充说明：

首先，由于在 *get_max_** 方法中需要以原始大规模关系集合 E 作为参数来进行处理，因此本文仍然采取 MapReduce 技术来高效提取不同策略下的关键节点. 这里我们以 MaxOut 策略为例介绍提取思路. 假设关系集合 E 中的每一条关系已根据社团结构初始化为$\langle v_i-C_i, v_j-C_j\rangle$格式. 在 Mapper 方法中需要判断两个节点是否被赋予不同的标签值. 因此，若不同则输出$\langle v_i-C_i,1\rangle$和$\langle v_j-C_j,1\rangle$，否则输出$\langle v_i-C_i,0\rangle$和$\langle v_j-C_j,0\rangle$. 在 Reducer 方法中则累计每一个节点相应的所有值并按照最终结果进行排序进而可以得到任意社团中 MaxOut 策略下的关键节点.

其次，由于在大规模网络中获取到的关键节点数量仍然有可能较多，因此在实际免疫的过程中可以只选取部分（一定比例）关键节点进行免疫同样可获得较好的抑制效果，而本文将会在第 6 节最后的实验部分给出不同比例下的抑制情况.

6 实验分析

6.1 实验环境

为了验证本文所提出的社团并行发现算法的高效可扩展性以及相应的 OSN 蠕虫抑制策略的有效性，我们搭建了分布式计算实验环境，如图 4 所示.

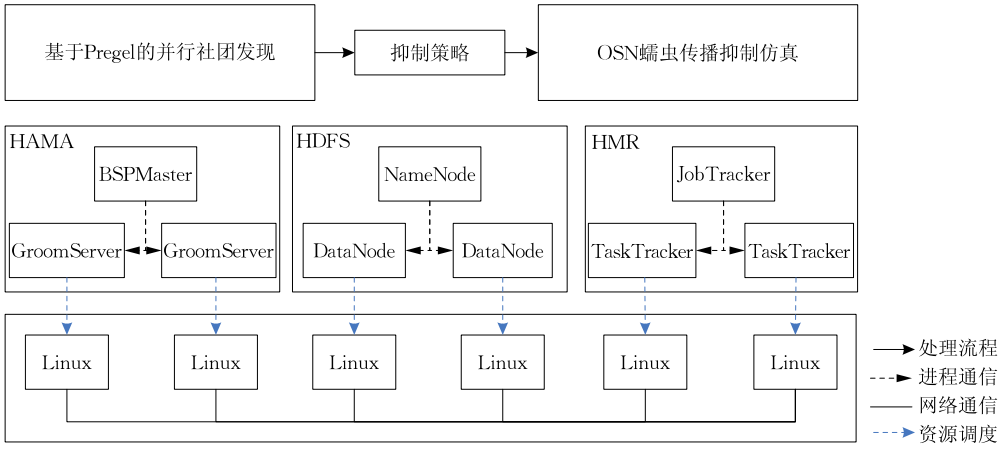


图 4 实验环境

其中，最底层的硬件节点集群是由 2 台高性能 Dell R910 服务器通过开源虚拟化软件 Oracle VirtualBox 来完成；中间层则是我们利用 Apache 开源项目来搭建的分布式软件计算及存储环境，主要包括了用于社团并行发现的 Pregel 技术的开源实现——Hama 项目以及用于蠕虫仿真的 MapReduce 技术的开源实现——Hadoop 项目(图中 HDFS 和 HMR 是该项目中的两个子项目)；而最上层则是本文所进行的实验，包括社团并行发现和 OSN 蠕虫传播抑制仿真.

本文在进行仿真实验时利用了作者之前提出的基于 MapReduce 的大规模 OSN 蠕虫传播仿真平台[27]，并通过扩展原有仿真功能，使其在提供抑制策略的条件下能够进行蠕虫传播抑制的仿真实验. 具体来说，本文在已获取到的 OSN 社团结构的基础上，增加了单个的节点免疫功能：即当已感染或者未感染节点接收到免疫或者修复补丁后，我们设定该节点从此不会再被蠕虫感染，并且其也会主动向周围邻居节点进行补丁下发，而当节点收到补丁后即被标记为免疫状态.

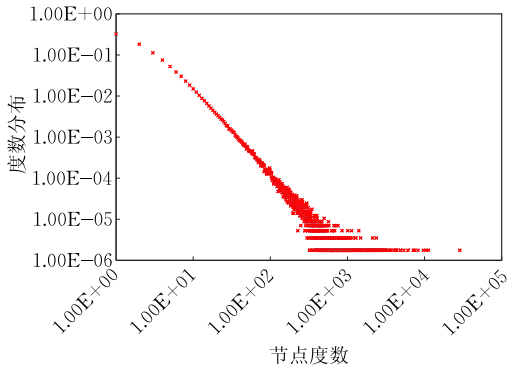
6.2　数据集

为了保证仿真实验的有效性和真实性，我们在实验过程中采用由文献[28]提供的两组真实且具有上百万节点规模的 OSN 用户关系数据，包括流行的视频分享社交网站 YouTube 以及图片分享社交网站 Flickr. 这两组关系数据的相关特征参数如表 1 所示. 可以看出，本文所选用数据集都具有较小特征路径长度和较大聚集系数①. 这些特点反映出了 OSN 网络所具有的小世界特性[1]，同时这些特性很大程度上有利于 OSN 蠕虫在整个网络中的快速传播. 具体来说，较小的特征路径意味着网络中任意两个节点之间拥有较小的距离，因此能够使得蠕虫在较短时间从初始感染节点传播到网络中的任意节点；而较大的聚集系数则意味着网络中的相邻节点的朋友关系拥有较大的重叠性，即“朋友的朋友仍然是朋友”，该特性提高了蠕虫感染用户好友的概率，进而加快了其在整个网络中的传播速度.

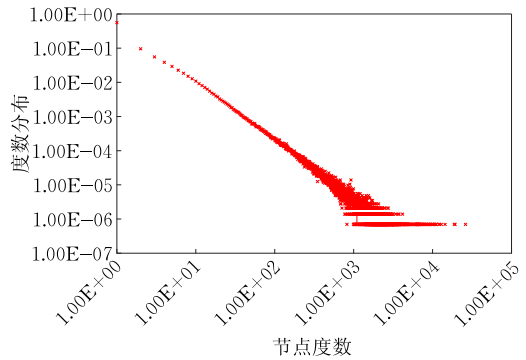
表 1　实验数据集

名称	节点数	边数	特征路径	聚合系数
YouTube	1157827	4945382	5.1	0.136
Flickr	1846198	22613981	5.7	0.313

此外，我们还分析了这两个数据集中节点度数分布情况. 在图 5 中，我们给出了该数据集中节点度数分布的双对数图(log-log plot). 可以看出两组数据集的节点度数均满足幂率分布的特点，该特点也反映了真实网络中的无标度特性[2]，即大部分节点只拥有较少邻居节点，而少部分节点拥有较多邻居节点. 该特性意味着如果 OSN 蠕虫能够成功感染后者，则其能够在短时间内感染网络中的大部分节点. 而这也解释了 HelloSamy 蠕虫为何能通过感染微博中明星账号的方式在 15 分钟内感染上万用户的原因.



(a) YouTube



(b) Flickr

图 5　数据集度数分布

6.3　实验结果及分析

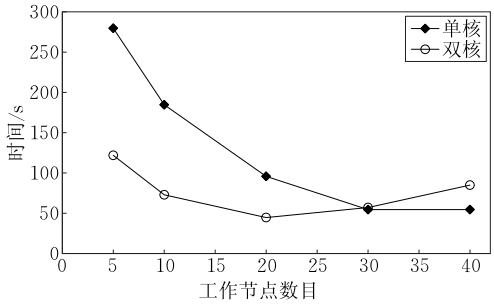
6.3.1　社团发现效果

为了检验 PLPA 算法处理大规模网络数据时的效率，本文针对该算法进行了可扩展性和有效性实验. 其中，可扩展性实验主要通过在不同节点(CPU 核数)的条件下观察算法运行时间来进行，而有效性实验则是通过观察算法在不同条件下的运行结果(即社团划分的真实性)来完成.

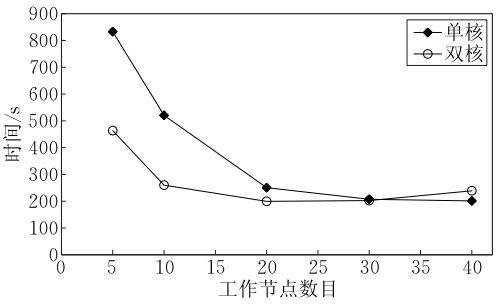
首先，在实际进行可扩展性实验过程中，一方面通过增加运算节点数来观察算法运行时间；另一方面还通过增加单个节点的 CPU 核数来观察其运行情况. 图 6 给出我们在两组数据集上的运行结果，其

① 关于特征路径、聚集系数的定义和计算方法可参见文献[1]里图 2 下方题注所描述的具体内容.

中实验结果是进行 10 次重复性实验后的平均值. 其中图 6(a)中显示了在 YouTube 数据集上的实际情况. 可以看出当投入的工作节点数从 5 增加到 30 时,单核运算时间已经减少至原来的 1/6,而双核运算时间减少至原来的 1/3. 从图 6(b)中也可看到节点数从 5 增加到 30 时,单核运算时间缩减至原来的 1/4,而双核情况下时间减少 1/2.



(a) YouTube



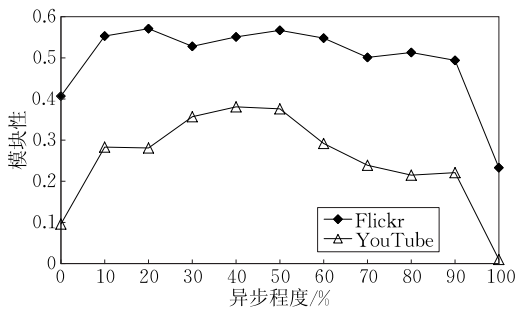
(b) Flickr

图 6 PLPA 算法可扩展性实验

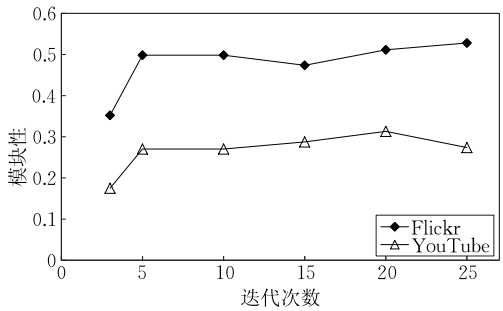
通过以上分析可以看出,本文提出的 PLPA 算法具有较好的可扩展性,即能够通过简单的增加运算节点数(CPU 核数)来加快算法的运行时间. 但需要说明的是,从图 6(a)和图 6(b)可以看出,对规模一定的数据,当投入节点数或 CPU 核数过多,例如两图中节点数超过 30 时的情况,反而会导致算法运行时间回升. 分析其主要原因在于,此时较多的时间将会浪费在分布式环境下的资源同步及节点间网络通信等非核心计算任务中. 而类似问题在其他分布式计算问题[27,29]中也均有出现. 因此,在实际计算过程中,我们需要根据待处理数据的规模来分配合理的计算节点数目,以达到最优效果,同时也应该避免因投入硬件设备过多而导致时间回升问题.

接下来,本文在上述可扩展性实验中获得最好效果的条件下,即在 30 个计算节点的环境中进行 PLPA 算法的有效性实验. 根据 PLPA 算法的特点知道,对算法结果可能产生影响的因素包括算法自身的异步程度以及实际迭代的次数. 因此,本文对这两个因素影响程度进行了具体的实验比较,最终结果如图 7 所示. 需要说明的是,为了度量算法的有效性,本文选取目前流行的模块性(Modularity)度量值[11],其相关计算公式可参见文献[11]. 在文献中,作者明确指出当模块性度量值大于 0.3 时,即可说明算法结果能够较好地反映网络的社团结构.

图 7(a)反映了不同的异步程度对算法结果的影响. 可以看出,当异步程度在(0,1)之间时,算法在 YouTube 数据集上运行结果具有较大的波动性,而在 Flickr 数据集上基本保持稳定. 其原因在于前者相对于后者本身的聚集系数较低,如 6.2 节表 1 所示,由于其自身真实的社团结构并不明显,进而造成不同的异步程度会对其造成较大的影响. 而对于聚集系数较大的 Flickr 数据集,由于其自身的社团结构比较明显,因此异步程度的取值对其影响效果不大. 此外,从图 7(a)中还可以看出,当异步程度为 0(同步策略)或者 1(完全异步策略)时,模块性度量值急剧下降. 因此,虽然在原始算法中并没有对异步程度值的选取进行任何限制,但通过实验可以反映出在实际应用中应当将其控制在(0,1)之间.



(a) 不同异步程度的影响



(b) 不同迭代次数的影响

图 7 PLPA 算法有效性实验

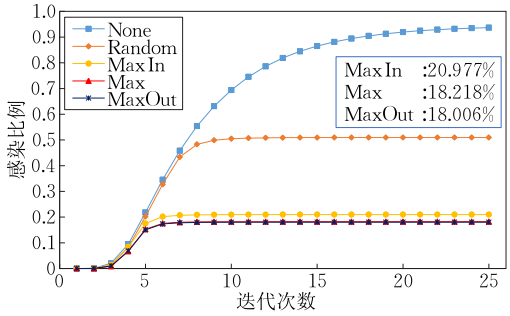
图 7(b)给出的是 PLPA 算法在不同的迭代次数下所产生的结果. 从图中可以看出,正如第 4.2 小节关于 PLPA 算法描述的那样,当迭代次数大于 5 时,算法所获取到的社团结构的模块性度量值基本保持不变,即在 YouTube 数据集上保持在 0.3 左

右，而在 Flickr 数据集上保持在 0.5 左右. 因此，我们通过实验再次证明了基于标签传播的社团发现算法只需要迭代较少的次数即可对大多数节点进行正确的社团标记，也说明了该算法能够以较快的速度完成社团发现任务.

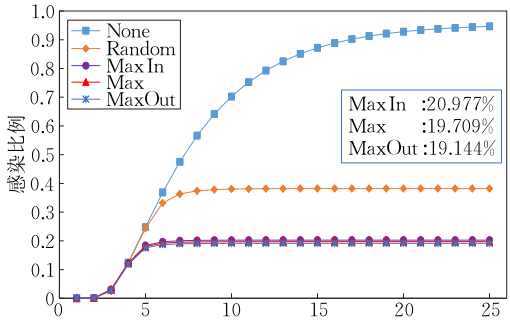
通过以上对可扩展性实验以及有效性实验的结果分析，可以看出，一方面本文通过提出 PLPA 算法解决了现有算法的可扩展性问题，使其能够在大规模网络中进行社团结构的发现；另一方面，由于 PLPA 算法在处理节点标签时，是面向节点为中心的计算过程，因此其本质上仍然是按照 LPA 算法来进行标签计算的，故其能够保留原始算法只需要较少迭代次数即可完成社团结构划分的优点[12,30]，这也为本文算法能够在大规模网络中高效完成社团发现提供了有力的实验证明.

6.3.2　OSN 蠕虫抑制效果

为了验证本文提出的基于社团并行发现的 OSN 蠕虫抑制方法的有效性，本文进行了两类蠕虫传播抑制仿真实验. 其中第 1 类实验是比较无抑制和采取随机抑制措施时的蠕虫防御效果(见图 8)；第 2 类实验是基于不同社团发现方法的抑制效果比较，这里本文选择和 Nguyen 等人在小规模网络中使用的 BGLL 算法以及 Zhu 等人使用的图划分方法相比较(见图 9).

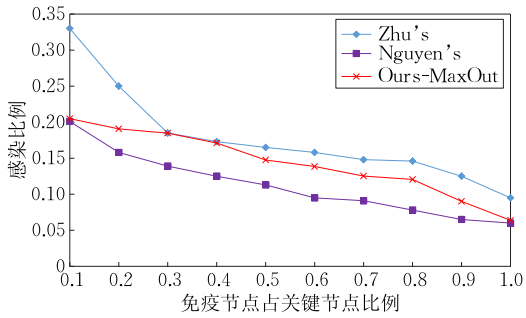


(a) YouTube

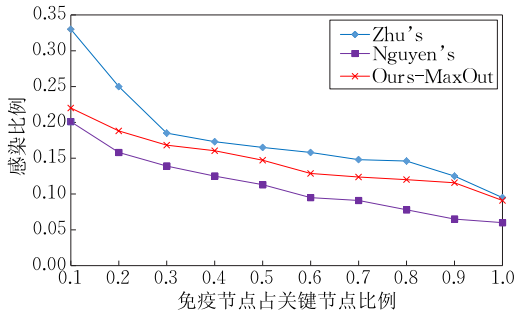


(b) Flickr

图 8　基于 PLPA 的 OSN 蠕虫抑制效果



(a) YouTube



(b) Flickr

图 9　OSN 蠕虫抑制效果对比

首先，图 8 给出了本文基于社团结构中关键节点免疫策略和无抑制及随机策略下的抑制效果比较. 需要说明的是，参照文献[7-8]中所提供的实验参数，在实验过程中，我们同样在蠕虫感染率超过 2%时开始启动蠕虫抑制过程，并且只选取了已发现关键节点中的 20%作为优先免疫节点. 从图 8(a)和(b)中可以看出，当不采取任何抑制措施时，OSN 蠕虫将会在 15 次迭代以后基本上感染网络中的所有节点. 而采取随机抑制的方式时，虽然能够起到一定的防御措施，但仍然会有大部分用户被感染(YouTube 数据集中 50%的节点，Flickr 数据集中 40%的节点). 而当我们采取基于社团结构中关键节点的抑制策略时，可以发现在传播过程迭代 5 次以后，被感染的用户数量基本保持在 20%左右. 进而可以反映出基于社团中关键节点的抑制方案能够对 OSN 蠕虫产生较好的防御效果.

此外，在图 8(a)和(b)中的蓝色框中，本文列出了 3 种关键节点选取策略下最终的具体抑制效果. 可以看出，在两个不同的数据集中，MaxOut 策略都取得了最好的抑制效果. 而对比本文所给出的 3 种策略的具体描述可以发现，在 MaxOut 策略中，通过选取和其他社团连接数最多的节点进行免疫可以保证尽可能多的社团接收到修复补丁，此外还可以确保本社团内的蠕虫不会轻易传播到其他社团中去. 而其他两种策略只是保证修复补丁能够以最快的速

度在本社团内传播，而无法抑制社团间蠕虫的快速传播.

最后，如图 9 所示，我们根据文献[8]所提供的实验数据，详细比较了 PLPA 算法(利用 MaxOut 策略)、Nguyen 等人使用的 BGLL 算法以及 Zhu 等人使用的图划分方法的基础上进行 OSN 蠕虫抑制的最终效果.为了保持和文献[7-8]的一致性，我们在蠕虫感染比例超过 2%时启动相应的抑制策略.此外，考虑到在大规模网络中所发现的社团数量及对应的关键节点数量仍然较多，这里我们通过选取不同比例的免疫节点来进行实际抑制效果的比较.

首先，从图 9(a)和(b)中给出的 3 种基于社团结构的 OSN 蠕虫抑制效果来看，只需要对部分关键节点进行免疫即可获得较好的抑制效果.例如，通过本文提出的抑制方法(利用 MaxOut 策略)仅免疫 20%的关键节点即可保证在两组实验中最终的感染比例不超过 20%，而这一结果也与前文图 8 所示的两组实验结果保持一致.此外，通过两组实验结果也可以看出，即使我们在只选取 10%的关键节点的条件下，最终的感染比例均不会超过 25%，因此从另一方面也说明了文本所提出的 OSN 蠕虫抑制方法的有效性和高效性.

其次，通过以上两组比较实验均可以发现，本文方法的最终抑制效果始终介于其他两种方法之间.造成该结果的主要原因在于 PLPA 算法本身所能够发现的社团结构的模块性度量值要小于 BGLL 算法所得到的度量值，但大于基于图划分方法的度量值.因此，该实验结果表明为了取得更好的 OSN 蠕虫抑制效果，除了增加社团发现算法的可扩展性以外，增强社团发现的准确度也是十分重要的，而这一部分内容是目前论文的主要弱点问题，同时也将是本文作者下一步研究的重点.

7 总结与展望

为了解决现有 OSN 蠕虫抑制方法无法有效应对大规模网络的问题，本文首先提出了基于 Pregel 模型的 PLPA 算法.该算法利用分布式计算模型能够有效处理大规模数据，同时还具备了基于标签传播方法的高效性，即通过少量迭代过程即可完成计算任务.之后，本文给出了 3 种不同的社团关键节点的提取策略，并通过对这 3 种策略下的关键节点进行优先免疫进而达到针对 OSN 蠕虫的快速抑制效果.最后，本文通过真实的数据集验证了算法的可扩展性以及抑制方法的有效性.

在接下来的工作中，我们一方面将继续深入研究能够获得较高模块性度量值的社团并行发现算法，同时还准备将 Pregel 模型应用到已有的重叠社团发现算法中，提出的重叠社团并行发现算法及 OSN 蠕虫抑制方法.拟通过选取重叠区内的免疫节点以望获得更好的 OSN 蠕虫抑制效果.

参 考 文 献

[1] Watts D J, Strogatz S H. Collective dynamics of 'Small world' networks. Nature, 1998, 393(6684): 440-442

[2] Barabasi A L, Albert R. Emergence of scaling in random networks. Science, 1990, 286(5439): 509-512

[3] Cao Y, Yegneswaran V, Porras P, Chen Y. PathCutter: Severing the self-propagation path of XSS JavaScript worms in web social networks//Proceedings of the 19th Network and Distributed System Security Symposium (NDSS). San Diego, USA, 2012

[4] Livshit B, Cui W. Spectator: Detection and containment of JavaScript worms//Proceedings of the USENIX Annual Technical Conference on Annual Technical Conference. Boston, Massachusetts, 2008: 335-248

[5] Xu W, Zhang F, Zhu S. Toward worm detection in online social networks//Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC). Austin, Texas, 2010: 11-20

[6] Sun F, Xu L, Su Z. Client-side detection of XSS worms by monitoring payload propagation//Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS). Saint-Malo, France, 2009: 539-554

[7] Zhu Z, Cao G, Zhu S, et al. A social network based patching scheme for worm containment in cellular networks//Proceedings of the IEEE INFOCOM. Rio de Janeiro, Brazil, 2009: 1476-1484

[8] Nguyen N P, Xuan Y, Thai M T. A novel method for worm containment on dynamic social networks//Proceedings of the 2010 Military Communications Conference (MILCOM). San Jose, USA, 2010: 2180-2185

[9] Nguyen N P, Dinh T N, Tokala S, Thai M T. Overlapping communities in dynamic networks: Their detection and mobile applications//Proceedings of the 17th Annual International Conference on Mobile Computing and Networking (MobiCom). Las Vegas, USA, 2011: 85-96

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

[11] Newman M E J. Fast algorithm for detecting community structure in networks. Physical Review E, 2004, 69: 066133

[12] Raghavan U N, Albert R, Kumara S. Near linear time algorithm to detect community structures in large-scale networks. Physical Review E, 2004, 76: 036106

[13] Palla G, Derenyi I, Farkas I, Vicsek T. Uncovering the overlapping community structure of complex networks in nature and society. Nature, 2005, 435: 814-818

[14] Shen H, Cheng H, Cai K, Hu M B. Detect overlapping and hierarchical community structure in networks. Physics A Statistical Mechanics and Its Applications, 2009, 388(8): 1706-1712

[15] Lee C, Reid F, McDaid A, Hurley N. Detecting highly overlapping community structure by greedy clique expansion// Proceedings of the 4th Workshop on Social Network Mining and Analysis. Washington, D.C., USA, 2010

[16] Gregory S. Finding overlapping communities in networks by label propagation. New Journal of Physics, 2010, 12(10): 1-26

[17] Blondel V D, Guillaume J L, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 2008, P10008

[18] Chen Y Y, Huang C, Zhai K. Scalable community detection algorithm with MapReduce. Communications of the ACM, 2009, 53: 359-366

[19] Yang S, Wang B, Zhao H, Wu B. Efficient dense structure mining using MapReduce//Proceedings of the 2009 IEEE International Conference on Data Mining Workshops. Washington, DC, USA, 2009: 332-337

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

[21] Malewicz G, Austern M H, Bik A J C, et al. Pregel: A system for large-scale graph processing//Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. Indianapolis, USA, 2010: 135-146

[22] Bao N T, Suzumura T. Towards highly scalable Pregel-based graph processing platform with X10//Proceedings of the International World Wide Web Conference (WWW). Rio de Janeiro, Brazil, 2013: 501-508

[23] Zhang Y, Wang J, Zhou L. Parallel community detection on large networks with propinquity dynamics//Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Paris, France, 2009: 997-1005

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

[25] Zhang J, Ge S. A parallel algorithm to find overlapping community structure in directed and weighted complex networks//Proceedings of the 2nd International Conference on Instrumentation & Measurement, Computer, Communication and Control. Harbin, China, 2012: 1561-1564

[26] Lancichinetti A, Fortunato S, Kertesz J. Detecting the overlapping and hierarchical community structure in complex networks. New Journal of Physics, 2009, 11: 033015

[27] He Liang, Feng Deng-Guo, Wang Rui, Su Pu-Rui, Ying Ling-Yun. MapReduce-based large-scale online social network worm simulation. Journal of Software, 2013, 24(7): 1666-1682(in Chinese)
(和亮，冯登国，王蕊，苏璞睿，应凌云. 基于 MapReduce 的大规模在线社交网络蠕虫仿真. 软件学报，2013，24(7)：1666-1682)

[28] Mislove A, Marcon M, Gummadi K P, et al. Measurement and analysis of online social networks//Proceedings of the 5th ACM/Usenix Internet Measurement Conference. San Diego, USA, 2007: 29-42

[29] Pan Wei, Li Zhan-Huai, Wu Sai, Chen Qun. Evaluating large graph processing in MapReduce based on message passing. Chinese Journal of Computers, 2011, 34(10): 1768-1784(in Chinese)
(潘巍，李战怀，伍赛，陈群. 基于消息传递机制的 MapReduce 图算法研究. 计算机学报，2011，34(10)：1768-1784)

[30] Zhang Jun-Li, Chang Yan-Li, Shi Wen. Overview on Label propagation algorithm and applications. Aplication Research of Computers, 2013, 30(1): 21-25(in Chinese)
(张俊丽，常艳丽，师文. 标签传播算法理论及其应用研究综述. 计算机应用研究，2013，30(1)：21-25)

HE Liang, born in 1985, assistant professor. His main research interests include system and network security and online social network worm.

FENG Deng-Guo, born in 1965, Ph. D., professor. His research interests include information security and cryptography.

SU Pu-Rui, born in 1976, Ph. D., professor. His research interests include system & network security and mobile security.

YING Ling-Yun, born in 1982, Ph. D., associate professor. His research interests include system & network security and P2P computing.

YANG Yi, born in 1982, Ph. D., assistant professor. His research interests include system & network security, mobile security and Android security.

Background

As one of the most popular communication services, online social networks (OSNs) have unfortunately caused new security threats, e. g. , OSN worms. Although several worm detection methods have been proposed by the security community, there still have other problems should be solved, such as worm containment. As it is impossible to patch all nodes at the same time when the worm propagation is detected, we need an efficient strategy to contain worms before all users are immune to them. Recently, the effectiveness of community detection based worm containment has been proved by security researchers. However, as current community detection methods all suffer from the problem of scalability, the aforementioned containment cannot sufficiently work well in large-scale OSNs.

This paper presents a novel worm containment which is based on parallel community detection. Particularly, we use the Pregel technology to reimplement the label propagation based community detection algorithm. And we also proposed various strategies to efficiently collect influential users in the communities and send patches to them first, and then propagate the patches by them. Finally, we implemented a prototype and conducted various experiments on it. The results show that our method can sufficiently contain the worm propagation in large-scale OSNs.

This work is supported by the National Basic Research Program (973 Program) of China under Grant No. 2012CB315804, the National Natural Science Foundation of China under Grant No. 61073179, the Major Research Plan of the National Natural Science Foundation of China under Grant No. 91118006 and Beijing Natural Science Foundation under Grant No. 4122086.