中国科学: 信息科学 2018 年 第 48 卷 第 3 期: 329–348
SCIENTIA SINICA Informationis
论文

支持可扩展的在线社交网络数据放置方法

周经亚[1,2,3*], 樊建席[1], 王进[1]

1. 苏州大学计算机科学与技术学院, 苏州 215006
2. 软件新技术与产业化协同创新中心, 南京 210093
3. 江苏省计算机信息处理技术重点实验室, 苏州 215006
* 通信作者. E-mail: jy_zhou@suda.edu.cn

收稿日期: 2017–03–28; 接受日期: 2017–10–24; 网络出版日期: 2018–02–13

国家自然科学基金 (批准号: 61502328, 61572337, 61672370)、江苏省产学研联合创新资金前瞻性研究 (批准号: BY2014059-02)、江苏省高校自然科学研究基金 (批准号: 15KJB520032) 和江苏省博士后科研资助计划 (批准号: 1701173B) 资助项目

摘要 在线社交网络的兴起吸引了越来越多用户的加入, 面对数以亿计的用户量, 如何以一种可扩展的方式存储用户数据已成为社交服务提供商和学术界共同关注的热点问题. 目前广泛采用的分布式键值存储通过 Hash 方法将用户数据随机放置在不同的存储服务器上, 这种方法会导致数据中心内部巨大的通信量, 不利于社交网络的扩展. 本文针对社交网络中用户交互的特点, 提出一种社交图划分与数据复制相结合的数据放置方法; 进一步考虑数据中心网络拓扑, 针对具体拓扑结构, 设计数据放置算法, 并分别讨论了算法对社交网络规模的增量调整以及分布式实现. 在真实数据集上的比较实验结果表明本文所提出的算法能够有效降低社交网络数据中心内部通信量, 增强其可扩展性.

关键词 在线社交网络, 数据放置, 数据中心网络, 社交局部性, 位置局部性

1 引言

在线社交网络 (online social networks, OSNs) 将人们的社交活动投射到虚拟网络空间, 其便捷的操作和没有时间空间限制的特点吸引了越来越多用户的加入. Facebook 月度用户数从 2012 年的 10 亿增长到 2016 年的 17 亿, 微信用户也从 1 亿增长到 6 亿. 面对如此庞大的用户群和较快的增长速度, 传统集中式存储显得无能为力, 分布式键值存储如 HDFS [1], Dynamo [2] 和 Cassandra [3] 等, 将用户数据通过 Hash 方式分散存放到数据中心内大量存储服务器上, 以解决大数据存储问题. 然而对于在线社交网络而言, 用户在交互过程中将频繁地访问彼此的数据, 如用户经常浏览好友发布的信息并对此适时地发表一些评论, 这些操作都涉及到对社交用户数据的读写. Hash 方式通常使得任意用户及其好友的数据分散存储在多台服务器上, 用户交互操作需要服务器间通信, 这势必导致数据中心内通信量

引用格式: 周经亚, 樊建席, 王进. 支持可扩展的在线社交网络数据放置方法. 中国科学: 信息科学, 2018, 48: 329–348, doi: 10.1360/N112017-00064
Zhou J Y, Fan J X, Wang J. Data placement approach for scalable online social networks (in Chinese). Sci Sin Inform, 2018, 48: 329–348, doi: 10.1360/N112017-00064

急剧增加. 早在 2013 年, Facebook 就曾透露其数据中心内部通信量已经达到外部通信量的 1000 倍[1)], 如此高的通信量势必占用数据中心内大量宝贵的带宽资源, 从而严重影响社交网络的扩展性.

之所以产生这一结果, 主要原因在于 Hash 方式的随机性并不适用于社交网络, 它没有考虑用户间的交互关系. 若将经常交互的用户数据放在一起, 则可避免大量不必要的通信, 社交局部性就是用来反映这些用户的数据能够放在一起的程度. 为了提高社交局部性, 有研究提出利用图分割算法 METIS [4] 或社区发现算法 [5] 将代表社交网络的社交图划分为多个子图, 每个子图中的用户存储在一起. 例如, 文献 [5] 通过对从 Facebook 采集的数据集分析发现社交网络具有自相似性的特征, 即根据用户交互行为构建的社区可以自组织为结构相似、规模更大的社区, 进而提出基于自相似性的数据划分算法, 但是该算法并没有考虑数据复制, 其社交局部性提升有限.

文献 [6∼8] 提出通过创建数据副本来保证访问数据与访问者自身数据存放在一处. SPAR [6] 将用户所有邻居的数据在该用户的主服务器上创建副本, 从而避免跨服务器读操作产生的通信量, 但是当服务器规模在 500 台时, SPAR 会为每个用户平均创建至少 20 个副本, 副本一致性维护将带来大量跨服务器写操作通信量. 由此可见, 单纯通过复制手段提高社交局部性并不一定能够很好地实现对通信量的优化. Tran 等 [7] 在更新代价固定前提下提出一种社交敏感的复制方法 S-clone, 其核心思想是将用户数据复制到存放其邻居最多的服务器上. 但是该方法并没有区分读操作和写操作, 当数据存放在较少读的服务器上时, 频繁的更新仍然会带来较高的写通信量. 针对这一情况, Liu 等 [8] 提出一种有选择的数据复制方法 SD^3, 每次总选择读频度高、更新频度低的用户创建数据副本, 可以在一定程度上降低复制带来的一致性维护通信量. Jiao 等 [9] 则将数据放置问题分解为两个子问题: 已知从属副本放置, 求最优主副本放置和已知主副本放置, 求从属副本放置, 通过对两个子问题的交替求解来优化包括通信量在内的多项指标.

近期研究 [10∼13] 提出将上述两种手段结合起来使用. Tran 等 [10] 同时考虑通信量与负载均衡, 将数据放置问题模型化为一个多目标优化问题, 提出一种基于遗传算法的划分方法, 并在划分基础上提出利用复制进一步降低通信量, 但由于遗传算法求解复制问题的解空间太大, 作者并未给出求解算法. Yu 等 [11, 12] 利用超图描述多个用户之间的交互行为, 将问题转化为超图分割问题, 并提出数据放置策略 ADP, 在超图分割基础上利用迭代方法为用户创建副本. 在上述文献中, 复制是在用户数据划分完成之后执行, 因而复制优化的效果受限于数据划分结果. Tang 等 [13] 研究副本复制对服务器间通信量的影响, 并据此提出一种轻量级的优化通信量的复制算法 TOPR, TOPR 通过交替执行数据划分和数据复制进一步优化通信量. 然而, 这两种手段并非同时执行, 而是以各自独立的方式执行, 各自的优化结果会相互影响, 难以达到理想优化效果. 此外, Jiao 等 [14] 提出一种贪心算法 cosplay, 通过不断地交换主从副本位置来降低通信代价, 然而 cosplay 针对多数据中心环境, 优化目标为跨数据中心通信量和响应时延, 由于数据中心弹性可扩展, 放置数据时可以不考虑数据中心间负载均衡问题.

针对上述问题, 本文提出一种支持可扩展的数据放置方法 (scalable data placement, SDP). SDP 同时考虑社交局部性和位置局部性, 对于前者的优化, 本文将问题转化为带重叠的社区划分问题, 社区与服务器对应, 每个社区内所有用户数据存储于一台服务器, 重叠区域用户则会在多台服务器创建多个副本, 从而实现数据划分与复制同步优化. 对于带重叠的社区划分问题, 近年来有不少研究 [15∼19] 关注并提出求解方法. 基于标签传播思想, 文献 [15] 提出重叠社区划分算法 COPRA, 它赋予每个节点多个标签, 并设定标签传播的概率接受准则, 但社区划分结果受制于标签设定参数, 随机选取标签会影响算法收敛性, 从而导致划分结果的稳定性差. 基于种子扩张思想, 文献 [16] 定义适应度函数, 并随机选取若干用户作为初始种子节点, 由这些种子节点开始不断吸纳新成员以扩张社区规模, 使得适应

1) https://gigaom.com/2013/06/19/meet-facebooks-new-network-architecture-its-a-fabric/.

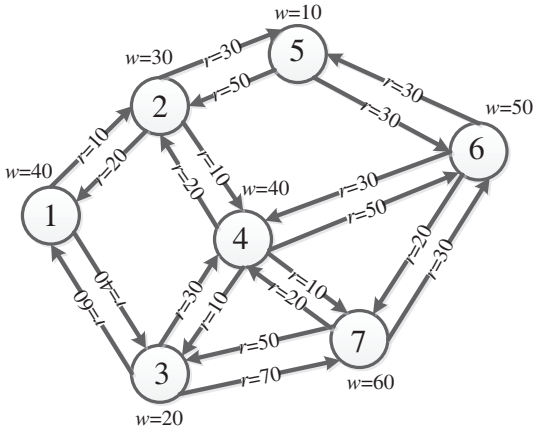

图 1 社交图示例

Figure 1 An example of social graph

度函数值不断得到优化, 直到网络被所有社区覆盖即网络中每个用户至少归属一个社区. 可见基于种子扩张的方法避免了标签数目设定对划分结果的影响, 然而, 采取随机方法选择初始种子节点仍然会导致社区划分结果不稳定. 为此, 文献 [17] 将初始种子节点的选择由随机产生改为极大连通子图, 文献 [18] 则基于全局信息找出核心节点作为初始社区. 文献 [19] 提出基于模块度的划分算法, 算法思想是采用平衡二叉树不断更新节点间模块度并进行社区合并, 算法先执行非重叠社区划分, 在此基础上再进行重叠社区识别, 虽然识别时间得到了优化, 但重叠社区识别依赖于非重叠社区划分结果, 稳定性得不到保障.

上述文献所提出的划分算法主要针对复杂网络, 通常被抽象为一个无向无权图, 图中社区被定义为高内聚的一组节点的集合, 集合内节点间彼此紧密互连, 与集合外节点连接稀疏. 这些方法均不适用于本文所研究的问题, 因为社交网络在本文中被定义为一个有向带权图, 不仅考虑网络节点间交互关系还考虑交互方向和交互产生的读写通信量 (图 1 中有向边和节点的权值).

网络通信量不仅取决于每次访问所涉及的服务器数目, 还取决于服务器在网络中的位置, 通信距离越远, 经过的网络设备越多, 通信量越高, 位置局部性用来衡量每次访问所涉及的服务器间的相近程度. 目前工作普遍缺乏考虑用户数据在具体网络拓扑中如何放置, 这使得位置局部性得不到有效保证, 从而影响对通信量的优化. 于是, 本文将问题进一步转化为社区到服务器的二次分配问题. 通过对两个子问题的求解, 获得数据优化放置方案, 降低数据通信量, 保持服务器间负载均衡, 增强在线社交网络扩展性.

2 模型框架

本节首先用社交图对在线社交网络进行建模, 同时对社交网络常用的两种数据中心网络拓扑结构进行介绍, 然后结合二者分析数据中心通信量与数据放置的关系. 最后, 给出数据优化放置问题的形式化定义.

2.1 在线社交网络模型

在线社交网络可以用一幅社交图 $G = (V, E)$ 来表示, 如图 1 所示, 其中顶点集 V 是所有用户的

集合, 边集 E 是用户间交互关系的集合. 考虑到社交交互行为具有频度和方向性的特点, 本文用一条有向边 $e_{uv} \in E$ 表示用户 u 到 v 的交互, 并赋予其权值 r_{uv}, 表示用户 u 访问 v 的频度. 用户间访问行为会产生读操作, 例如 Facebook 用户 u 经常浏览好友 v 的页面, r_{uv} 则表示单位时间内 u 读 v 的次数. 社交网络中每个用户 u 同样被赋予一个权值 w_u, 表示其自身数据更新的频度, 例如用户发布新状态, 这样会产生写操作. 文献 [20] 根据采集的社交数据发现用户间 92% 的交互行为是访问, 即产生读操作. 为了简化模型, 对于社交网络中的写操作交互, 如 u 评论 v 的状态, 我们没有显式地定义 w_{uv}, 即 u 向 v 执行写交互的频度. 但是, 因为写交互可以被分解为 u 读 v 和 v 更新自身两个元操作, 所以本文所提模型能够包含这两种交互. 社交图中具有交互关系的用户彼此互为邻居, 将 u 的邻居集记为 $N(u)$.

目前, OSN 存储系统均采用单主副本多从属副本 (single-master-multi-slave) 方式存储和管理用户数据. 在该存储方式下, 每个用户有且仅有一个主副本, 其余均为从属副本, 存放主副本的服务器称为该用户 u 的主服务器, 记为 m_u, 而存放从属副本的服务器的集合用 s_u 表示. 用户的所有操作均通过其主服务器完成, 例如, 用户 u 登录后首先到其主服务器 m_u 获取数据, 当其访问用户 v 时, 若 v 其中一个副本存放在 m_u 上, 则可直接访问, 否则需要到存储有 v 的副本的服务器上读取数据, 即 $m_v \cup s_v$ 中的一个服务器上取数据, 从而产生读通信量. 当用户 u 更新自身数据时, 为了维护副本一致性, 也会产生从 m_u 到 s_u 中其他服务器的写通信量. 根据 m_u 和 s_u 可以得出存放在服务器 x 上的所有用户副本的集合:

$$\phi_x = \{u \in V \,|x \in m_u \cup s_u\}, \tag{1}$$

其中, 存放用户主副本的集合可表示为

$$M_x = \{u \in V \,|x = m_u\}. \tag{2}$$

通信量与访问请求数和被请求用户的数据大小有关, 为了简化模型, 用单位时间内访问请求量定义通信量. 根据以上分析, 从服务器 x 到 y 的通信量 C_{xy} 分别由读通信量和写通信量构成, 其中读通信量表示以 x 为主服务器的用户对服务器 y 上用户数据的读请求量; 写通信量表示以 x 为主服务器的用户对其在服务器 y 上的数据副本更新请求量. 由此可得

$$C_{xy} = \sum_{u \in M_x} \sum_{v \in \phi_y} r_{uv} + \sum_{u \in M_x} g(u,y) w_u, \tag{3}$$

其中, $g(u,y)$ 表示 u 是否有副本存放于服务器 y, 若有, 则 $g(u,y)$ 为 1, 否则为 0, 即

$$g(u,y) = \begin{cases} 1, & u \in \phi_y, \\ 0, & u \notin \phi_y. \end{cases} \tag{4}$$

用 S 表示所有服务器集合, 进一步可得出服务器 x 输出的通信量:

$$C_x = \sum_{y \in S \setminus \{x\}} C_{xy}. \tag{5}$$

在引言的相关工作中指出, 通过提高社交局部性不一定能够实现对通信量的优化, 因为当利用复制改进社交局部性时有可能导致写通信量增加. 因此, 本文将社交局部性重新定义为用户数据放在一起能够减少读写通信量的程度, 并用服务器的输出通信量 C_x 度量. C_x 越小说明越多的邻居副本放在一起且产生的通信量越低, 从而达到越好的社交局部性.

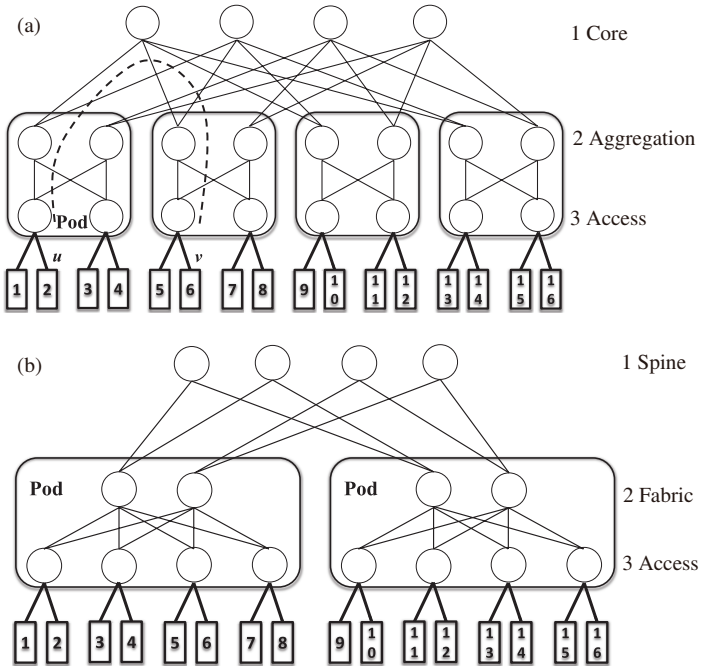

图 2 两种数据中心网络拓扑结构示意图

Figure 2 Two types of data center network topologies. (a) Fat-tree ($k = 4$); (b) Fabric ($k_s = 2,\ k_f = 2,\ k_a = 4,\ k = 4$)

2.2 数据中心网络拓扑结构

传统基于树型拓扑的网络, 由于单点失效以及对汇聚层和核心层网络设备要求高, 已不适用于构建大规模数据中心网络 [21]. Fat-tree [22] 对树型拓扑进行改进, 将接入交换机和汇聚交换机划分成若干 pod, 每个 pod 内两类交换机按照完全二分图互联. Fat-tree 中所有交换机均为普通商用交换机, 如图 2(a) 所示, 每个 k 口接入交换机接入 $k/2$ 台服务器, 共有 k 个 pod, 每个 pod 内接入交换机和汇聚交换机各 $k/2$ 个, 每个汇聚交换机与 $k/2$ 个核心交换机互联, 从而保证每个 pod 可以和全部 $k^2/4$ 个核心交换机互联. Fat-tree 容纳服务器数最高可达 $k^3/4$ 台, 并提供服务器之间无阻塞通信, 因而被广泛采用.

基于大量集群的 Facebook 采用 Clos 结构 [23] 构建其数据中心网络拓扑 [24], 然而其集群规模受限于集群交换机端口数量, 已不能很好适应日益增长的业务需求, 于是, Facebook 设计了下一代数据中心网络 Fabric[2), 其拓扑结构如图 2(b) 所示. 作为网络的基本组成单位, 每个 pod 包含 k_f 个 fabric 交换机和 k_a 个接入交换机, 其中前者采用光纤交换机提供上下行无阻塞通信, 后者为普通商用 k–口交换机. 为实现全网通信, Fabric 设置 k_f 个独立的 spine 交换机平面, 每个平面由 k_s 个 spine 交换机和所有 pod 中的一个 fabric 交换机互连而成. pod 和平面的组合构成了模块化的网络拓扑, 最多可容纳 $pk_a(k - k_f)$ 台服务器, 其中 p 为 pod 数目. 在 Facebook 的数据中心中 $k_a = k = 48$, $k_f = 4$, $k_s \leqslant 48$, 而当 $p = 48$ 时, 服务器数目可达 10 万台, 网络聚合带宽可达 PB 级.

本文将数据中心通信量定义为经过数据中心网络中各层交换机的通信量之和. 由此可见, 网络通信量与拓扑结构紧密相关, 通信双方在拓扑中的位置决定了通信需要经过的路径, 路径越长 (即经过的交

2) https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network.

换机越多) 则由此产生的通信量越高. 鉴于 Fat-tree 的广泛使用和 Fabric 在 Facebook 的成功应用, 本文主要针对上述两种网络拓扑结构研究数据放置问题, 其研究方法可进一步推广到其他类型的数据中心网络. 为了获得经过每层交换机的通信量, 首先需要确定通信双方在拓扑中的位置, 假设服务器已按图 2 所示顺序依次编号, 服务器位置很容易通过编号确定. 再根据编号可推导出服务器间的通信路径, 这里用最短路径表示, 以及该路径上经过的每层交换机数目. 例如, 图 2(a) 中服务器 u 和 v 的编号分别为 2 和 6, 由 $\lfloor 2\times 2/4\rfloor \neq \lfloor 2\times 6/4\rfloor$, 可知 u 和 v 没有连入同一交换机, 又由 $\lfloor 4\times 2/4^2\rfloor \neq \lfloor 4\times 6/4^2\rfloor$, 可知 u 和 v 不在同一 pod 内, 故 u 和 v 间通信需要经过 1 个核心层交换机、2 个汇聚层交换机和 2 个接入层交换机. 因此, 做如下定义:

$$
l_{\text{fattree-1}}=\begin{cases}1, & \left\lfloor\dfrac{2x}{k}\right\rfloor\neq\left\lfloor\dfrac{2y}{k}\right\rfloor\wedge\left\lfloor\dfrac{4x}{k^2}\right\rfloor\neq\left\lfloor\dfrac{4y}{k^2}\right\rfloor,\\ 0, & \text{otherwise;}\end{cases}
$$

$$
l_{\text{fattree-2}}=\begin{cases}1, & \left\lfloor\dfrac{2x}{k}\right\rfloor\neq\left\lfloor\dfrac{2y}{k}\right\rfloor\wedge\left\lfloor\dfrac{4x}{k^2}\right\rfloor=\left\lfloor\dfrac{4y}{k^2}\right\rfloor,\\ 2, & \left\lfloor\dfrac{2x}{k}\right\rfloor\neq\left\lfloor\dfrac{2y}{k}\right\rfloor\wedge\left\lfloor\dfrac{4x}{k^2}\right\rfloor\neq\left\lfloor\dfrac{4y}{k^2}\right\rfloor,\\ 0, & \text{otherwise;}\end{cases}
\tag{6}
$$

$$
l_{\text{fattree-3}}=\begin{cases}1, & \left\lfloor\dfrac{2x}{k}\right\rfloor=\left\lfloor\dfrac{2y}{k}\right\rfloor,\\ 2, & \left\lfloor\dfrac{2x}{k}\right\rfloor\neq\left\lfloor\dfrac{2y}{k}\right\rfloor,\\ 0, & \text{otherwise.}\end{cases}
$$

$$
l_{\text{fabric-1}}=\begin{cases}1, & \left\lfloor\dfrac{x}{k-k_f}\right\rfloor\neq\left\lfloor\dfrac{y}{k-k_f}\right\rfloor\wedge\left\lfloor\dfrac{x}{k_a(k-k_f)}\right\rfloor\neq\left\lfloor\dfrac{y}{k_a(k-k_f)}\right\rfloor,\\ 0, & \text{otherwise;}\end{cases}
$$

$$
l_{\text{fabric-2}}=\begin{cases}1, & \left\lfloor\dfrac{x}{k-k_f}\right\rfloor\neq\left\lfloor\dfrac{y}{k-k_f}\right\rfloor\wedge\left\lfloor\dfrac{x}{k_a(k-k_f)}\right\rfloor=\left\lfloor\dfrac{y}{k_a(k-k_f)}\right\rfloor,\\ 2, & \left\lfloor\dfrac{x}{k-k_f}\right\rfloor\neq\left\lfloor\dfrac{y}{k-k_f}\right\rfloor\wedge\left\lfloor\dfrac{x}{k_a(k-k_f)}\right\rfloor\neq\left\lfloor\dfrac{y}{k_a(k-k_f)}\right\rfloor,\\ 0, & \text{otherwise;}\end{cases}
\tag{7}
$$

$$
l_{\text{fabric-3}}=\begin{cases}1, & \left\lfloor\dfrac{x}{k-k_f}\right\rfloor=\left\lfloor\dfrac{y}{k-k_f}\right\rfloor,\\ 2, & \left\lfloor\dfrac{x}{k-k_f}\right\rfloor\neq\left\lfloor\dfrac{y}{k-k_f}\right\rfloor,\\ 0, & \text{otherwise.}\end{cases}
$$

上述表达式针对两种拓扑结构分别计算任意一对服务器 x 和 y 间通信在每层经过的交换机数目, 据此各层通信量可计算如下:

$$
\begin{cases}C_{\text{fattree-}i}=\displaystyle\sum_{x\in S}\sum_{y\in S}C_{xy}l_{\text{fattree-}i},\\ C_{\text{fabric-}i}=\displaystyle\sum_{x\in S}\sum_{y\in S}C_{xy}l_{\text{fabric-}i}.\end{cases}
\tag{8}
$$

表 1 本文所用变量与说明

Table 1 The variables and meanings used in this paper

Variable	Meaning
$G=(V,E)$	Social graph, where V is the vertex set and E is the edge set
r_{uv}, w_{uv}	The read rate and write rate from u to v
w_u	The write rate of u
$N(u)$	The set of u's neighbors
ϕ_x	The set of replicas stored at server x
m_u	u's master server
s_u	The set of u's slave servers
M_x	The set of users whose master server is x
C_{xy}	The traffic from x to y
$g(u,y)$	Whether u's replica is stored at server y
S	The server set
C_x	The traffic transmitted from server x
k	The number of switch ports
k_f, k_a, k_s	The switch numbers on fabric, access and spine levels, respectively
p	The number of pods in Fabric
$l_{\text{fattree-}i}$, $l_{\text{fabric-}i}$	The numbers of ith level switches between x and y in both topologies
$C_{\text{fattree-}i}$, $C_{\text{fabric-}i}$	The traffic on ith level in both topologies
load_x	The workload of server x
gini	Load balancing level
f_u	u's activity
$\Delta C(v,\text{com})$	The traffic change after v joins community com
θ	System redundancy
n	The number of communities returned by Algorithm 2
d	The average number of neighbors owned by a user

若通信主要发生在位置临近 (通信路径短) 的服务器间, 则可将通信量限制在底层交换机, 避免进一步扩大, 从而保持良好的位置局部性.

2.3 问题描述

负载均衡是除通信量之外影响可扩展性的另一个重要因素. 服务器存储的用户数据副本数目在很大程度上决定了其访问负载大小, 因此将服务器负载 load_x 定义为存储的副本数. 负载均衡度量方法有很多种, 如方差、Gini 系数等, 本文用 Gini 系数衡量负载均衡水平 [10], 定义如下:

$$\text{gini}=\frac{\sum_{x\in S}\sum_{y\in S}|\text{load}_x-\text{load}_y|}{2\,|S|\sum_{x\in S}\text{load}_x}. \tag{9}$$

之所以选择 Gini 系数, 除了其取值独立于系统规模之外, 还因为其取值范围为 [0, 1], 数值含义更直观, 0 表示绝对均衡而 1 表示最不均衡. 本文中所用的变量可参考表 1.

根据以上分析, 本文将问题描述如下: 给定社交网络 G 及其用户间交互情况 $\{r_{uv},w_u|u,v\in V\}$, 和数据中心拓扑 (Fat-tree 或 Fabric) 及服务器集 S, 找出最佳数据放置方案 $\{\phi_x|x\in S\}$, 优化数据中

心通信量, 同时保持良好负载均衡. 形式化定义为

$$\text{Minimize} \sum_{i=1}^{3} C_{\text{fattree-}i} \quad \text{or} \quad \sum_{i=1}^{3} C_{\text{fabric-}i}, \tag{10}$$

$$\begin{aligned} \text{s.t.} \quad & \text{(i)}\ |m_u| = 1, \\ & \text{(ii)}\ |m_u \cap s_u| = 0, \\ & \text{(iii)}\ \sum_{u \in V} |m_u \cup s_u| \leqslant |V|\theta^*, \\ & \text{(iv)}\ \text{gini} \leqslant \text{gini}^*, \end{aligned}$$

约束 (i) 保证每个用户均有一个主副本; 约束 (ii) 保证用户在一台服务器上最多有一个副本; 约束 (iii) 保证系统中存储的总用户副本数量不超过 $|V|\theta^*$, 其中 θ^* 为系统冗余度阈值, 即社交用户的平均副本数的上限; 约束 (iv) 保证负载均衡不超过给定阈值 gini^*.

3 数据放置算法

针对上述问题, 本文从社交局部性和位置局部性两方面入手. 通过优化社交局部性可以将经常发生交互的用户数据放置在一起, 从而降低通信量. 在介绍本文算法前, 先比较几种已有的数据放置算法, 如图 3 所示, 算法需要将图 1 所示的社交网络中用户分成两部分放在两台服务器上, 考虑负载均衡, 设 $\text{gini}^* = 0.1$. 图 3(a) 显示的是 Hash 算法划分的结果, 由于没有做任何优化, 其通信量高达 470. SPAR 算法先对社交图进行划分, 然后对部分用户数据进行复制, 使得用户及其所有邻居的数据放在一起, 故其读通信量为零, 但同时增加了更新产生的写通信量 220, 如图 3(b) 所示. ADP 算法利用超图划分社交网络可以获得更低的读通信量, 由于负载均衡的限制, 无论如何划分, 只能将用户划分成分别包含 3 和 4 个用户的两个部分, ADP 在划分基础上通过复制方法优化流量, 如图 3(c) 所示, 最终得到的通信量为 200. 上述两种算法都是结合社交图划分和数据复制对通信量进行优化, 从例子中可见, 相比于 Hash 算法确实可以在保障负载均衡的前提下显著降低通信量. 但是上述算法选择的结合方式是先划分后复制, 划分算法以降低读通信量为目标, 基于划分结果进行复制, 复制以优化写通信量为目标, 而复制本身会改变读通信量, 故这种异步优化很难达到最优效果. 如果划分和复制同步进行则可以取得更好的优化效果, 如图 3(d) 所示, 在数据放置时不仅考虑划分产生的读通信量, 也考虑复制产生的写通信量, 可以获得较之前算法更低的通信量.

为了实现同步优化, 将数据放置问题首先分解为带重叠的社区划分子问题, 把用户划分成若干社区, 以降低社区间通信量为目标. 在划分过程中考虑到存在一部分用户与多个社区的邻居节点交互频繁, 单纯分割无法降低通信量, 所以允许这部分用户归属多个社区, 并在相应社区创建副本. 该方法使得重叠社区中的用户分别属于多个社区, 自然映射为在每个归属的社区有一个副本, 对于具体某个用户而言, 为其创建的副本数量取决于社区划分中其归属的社区数, 而归属何社区则依据加入该社区是否对降低通信量有贡献. 重叠社区的出现将划分和复制同步进行.

获得用户社区划分之后, 还需要将社区用户的数据存放到服务器上, 问题被进一步分解为社区到服务器的二次分配子问题. 为了优化位置局部性, 把彼此间通信量较大的社区分配到距离较近的服务器, 以降低数据中心内的通信量. 根据 Fat-tree 和 Fabric 分层互连结构的特点, 提出逐层映射求解的思想, 自顶向下对服务器分组并与社区建立一对一映射, 最终实现社区到服务器的配对.

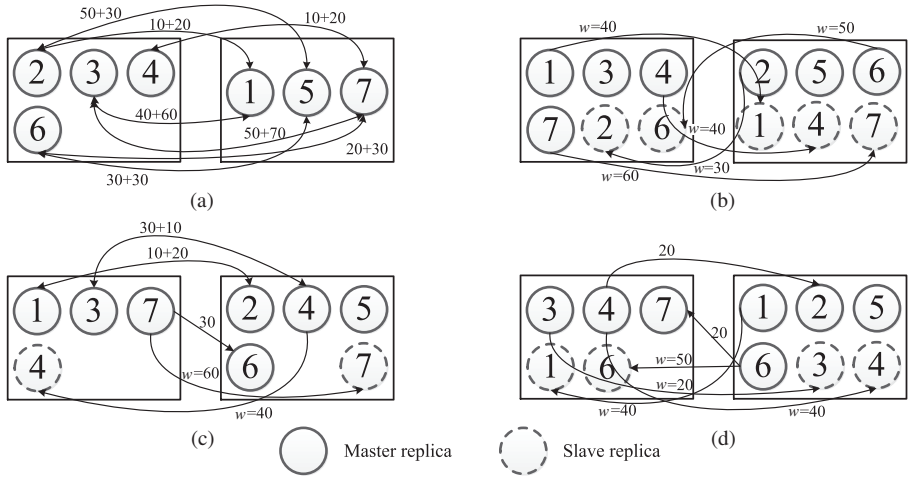

图 3 几种数据放置算法示例

Figure 3 The examples of several data placement algorithms. (a) Hash traffic=470 (read=470, write=0, gini=0.036); (b) SPAR traffic=200 (read=0, write=220, gini=0); (c) ADP traffic=200 (read=100, write=100, gini=0.028); (d) simultaneous optimization traffic=190 (read=40, write=150, gini=0.023)

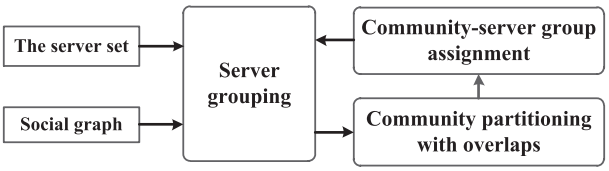

图 4 SDP 的执行逻辑

Figure 4 The execution logic of SDP

3.1 SDP 算法设计

根据上述问题求解思想, 本文提出一种新的数据放置方法 SDP, 其执行逻辑如图 4 所示, 伪代码如算法 1 描述. 首先, 将数据中心内服务器集合按照互连层次分组, 第 1 轮所有接入同一个 pod 的服务器为一组, Fat-tree 和 Fabric 分别划分出 k 和 p 个分组 (第 1 行), 接着将社交图划分出对应同等数量的社区 (第 3 行), 社区划分由算法 parOverlapComm 实现, 其返回值为社区集合和社区间通信量的集合. 在社区和服务器分组之间建立一对一映射, 这些服务器组间通信经过第 1 层 (顶层) 交换机, 无论采用何种映射, 经过第 1 层的通信量 ($C_{\text{fattree-1}}$, $C_{\text{fabric-1}}$) 均不会发生变化, 故采用 Hash 映射 (第 4∼7 行). 在接下来的每一轮, 将依次根据上一轮的结果进行更进一步的划分和映射. 第 2 轮对第 1 轮的每一个分组再分组, 接入同一个接入交换机的服务器归为一组, Fat-tree 和 Fabric 分别划分出 $k/2$ 和 k_a 个分组 (第 10 行), 对于第 1 轮中划分出的每个社区, 根据社交网络的自相似性 [5], 同样可以再划分出 $k/2$ 和 k_a 个子社区. 经过第 2 层的通信量分别为 $2C_{\text{fattree-1}}+C_{\text{fattree-2}}$ 和 $2C_{\text{fabric-1}}+C_{\text{fabric-2}}$. 网络拓扑共 3 层, 所以算法需执行 3 轮, 在最后一轮中, 每个服务器作为一个独立分组, 上一轮每一分组中分别包含 $k/2$ 和 $k-k_f$ 个服务器 (第 17 行). 最终, 产生的社区–服务器映射即为数据放置方案 (第 18∼20 行), 经过第 3 层的通信量分别为 $2C_{\text{fattree-1}}+2C_{\text{fattree-2}}+C_{\text{fattree-3}}$ 和 $2C_{\text{fabric-1}}+2C_{\text{fabric-2}}+C_{\text{fabric-3}}$.

算法 1 SDP

Input: $G(V,E)$, S, gini*;
Output: $\{\phi_x\}$, $C_{\text{fattree-}i}$ or $C_{\text{fabric-}i}$;
将 S 分成 k or p 个分组, 记为 sc1_i ; //分别对应两种拓扑结构
根据式 (6) 或 (7) 计算 $l_{\text{fattree-}i}$ or $l_{\text{fabric-}i}$, $i=1,2,3$;
$(\text{Com1}_i, C_{ij}) \leftarrow \text{parOverlapComm}(G, k \text{ or } p, \text{gini}^*)$;
根据式 (8) 计算 $C_{\text{fattree-1}}$ or $C_{\text{fabric-1}}$;
for $i \leftarrow 1$ to k or p **do**
    $\phi 1_i \leftarrow \text{hash}(\text{Com1}_i)$; //分区与服务器分组一对一映射
end for
for $i \leftarrow 1$ to k or p **do**
    将 sc1_i 分成 $k/2$ or k_a 个分组, 记为 sc2_j;
    $(\text{Com2}_j, C_{jm}) \leftarrow \text{parOverlapComm}(\text{Com1}_i, k/2 \text{ or } k_a, \text{gini}^*)$;
    for $j \leftarrow 1$ to $k/2$ or k_a **do**
        $\phi 2_j \leftarrow \text{hash}(\text{Com2}_j)$;
    end for
    根据式 (8) 计算 $C_{\text{fattree-2}}$ or $C_{\text{fabric-2}}$;
    for $j \leftarrow 1$ to $k/2$ or k_a **do**
        将 sc2_j 分成 $k/2$ or $k-k_f$ 个服务器, 记为 x;
        $(\text{Com}_x, C_{xy}) \leftarrow \text{parOverlapComm}(\text{Com2}_j, k/2 \text{ or } k-k_f, \text{gini}^*)$;
        for $x \leftarrow 1$ to $k/2$ or $k-k_f$ **do**
            $\phi_x \leftarrow \text{hash}(\text{Com}_x)$;
        end for
        根据式 (8) 计算 $C_{\text{fattree-3}}$ or $C_{\text{fabric-3}}$;
    end for
end for
return $\{\phi_x\}$, $C_{\text{fattree-}i}$ or $C_{\text{fabric-}i}$.

3.2 parOverlapComm 算法设计

为了达到优化通信量的目的, 此处将社区定义为彼此间通信量大的一组节点的集合. 本文借鉴种子扩张的思想设计求解算法. 首先从社交图中选取若干用户作为种子节点, 从种子的邻居中选取用户作为社区初始成员, 然后逐步以社区的邻居为对象扩张社区. 为了选出种子节点, 定义用户活跃度 f_u 作为筛选标准:

$$f_u = \sum_{v \in N(u)} r_{uv}. \tag{11}$$

用户 u 的活跃度取其访问所有邻居的频度之和, 数值越高说明 u 的社交活动越活跃. 社区划分的目标是尽可能降低社区间的通信量, 当判断用户 v 是否可以加入社区 com 时, 需要比较 v 加入前后通信量的变化. 根据 v 是否已归属其他一个或多个社区, 分情况讨论如下.

(1) v 尚未归属任何社区. 如图 5(a) 所示, 假定没有归属社区的用户均属于一个虚拟社区 vcom. 对于用户 v, 加入社区 com 前, 与社区 com 相关的通信量可表示为 $\sum_{u \in \text{com} \cap N(v)} (r_{vu} + r_{uv})$, 与 vcom 的通信量为零. 当 v 加入社区 com 后, 则前者变为零, 而后者则变为 $\sum_{w \in \text{vcom} \cap N(v)} (r_{vw} + r_{wv})$. 所以, 通信量变化可表示为

$$\Delta C(v, \text{com}) = \sum_{u \in \text{com} \cap N(v)} (r_{vu} + r_{uv}) - \sum_{w \in \text{vcom} \cap N(v)} (r_{vw} + r_{wv}). \tag{12}$$

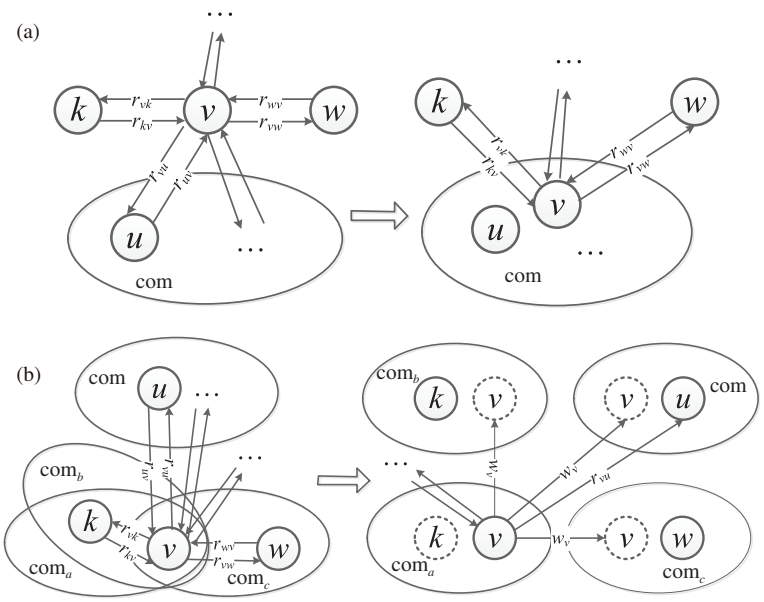

图 5 用户 v 加入社区 com 后通信量的变化

Figure 5 Traffic reduction after user v joins a community com. (a) User v has never been allocated to any community; (b) user v has been included in multiple communities

(2) v 已归属至少一个社区. 如图 5(b) 所示, v 已归属 com_a, com_b 和 com_c 3 个社区, 此时, v 以在社区 com 创建副本的形式加入该社区, 副本可以节省读通信量 $\sum_{u\in\mathrm{com}\cap N(v)} r_{uv}$, 同时也增加了更新通信量 w_v. 因此, 通信量变化为二者之差:

$$\Delta C(v,\mathrm{com}) = \sum_{u\in\mathrm{com}\cap N(v)} r_{uv} - w_v. \tag{13}$$

带重叠的社区划分方法由算法 2 parOverlapComm 描述. 首先, 计算输入图中所有用户的活跃度, 选出 top-n 个用户作为种子节点 (第 2, 3 行), 其中 n 为输入参数, 表示划分的社区数. 以每个种子节点用户作为社区的初始成员 (第 5 行), 对于每个社区 com_i, 找出与该社区成员交互的邻居的集合 $N(\mathrm{com}_i)$, 将其作为扩张社区的候选成员 (第 7, 8 行). 为了保证负载均衡, 在每次挑选候选成员之前, 先判断该社区是否过载, 即社区的成员数是否在允许的变动范围内 (第 9, 10 行), 其中 load 为当前社区的平均负载, 其初值与系统冗余度 θ 即平均副本数目有关, 此处设置为 $|V|\theta/n$, 表示社区容纳用户数目的平均值 (第 1 行). 每个社区完成扩张后, 更新一次 load 值 (第 16 行). 若未过载, 则继续判断通信量是否下降, 若是, 则加入社区 (第 11, 12 行). 每经过一轮对邻居集的筛选后, 若没有添加新邻居到社区, 则该社区不再继续扩张 (第 15 行). 最后, 根据划分结果计算社区间通信量, 并连同社区集一起作为返回值 (第 19, 20 行).

3.3 算法时间复杂度分析

首先分析 parOverlapComm 算法的时间复杂度, 假设社交网络 G 中用户的平均邻居数为 d, 计算所有用户活跃度 f_u (第 2 行) 需要的时间复杂度为 $O(d|V|)$, 其中 $|V|$ 表示 G 中用户数目, 选出 top-n 用户 (第 3 行) 的时间复杂度为 $O(n|V|)$. 社区包含的平均用户数即负载为 $|V|\theta/n$, 其中 θ 表示冗余度, 每加入一个成员用户都需要计算一次 $\Delta C(v,\mathrm{com}_i)$(第 11, 12 行), 计算复杂度为 $O(d)$, n 个社区所

算法 2 parOverlapComm

Input: $G(V,E)$, n, gini*;
Output: $\{\mathrm{Com}_i\}$, $\{C_{ij}\}$;
1: load $\leftarrow |V|\theta/n$;
2: 根据式 (11) 计算每个用户的活跃度 f_u;
3: 根据 f_u 值选出 top-n 用户;
4: **for** $u \in$ top-n 用户 **do**
5: u 初始化社区 com_i;
6: **do**
7: 找出 com_i 的所有邻居 $N(\mathrm{com}_i)$;
8: **for** $v \in N(\mathrm{com}_i)$ **do**
9: **if** $|\mathrm{com}_i| \geqslant \mathrm{load}(1+\mathrm{gini}^*)$ **then**
10: **break**;
11: **else if** $\Delta C(v, \mathrm{com}_i) > 0$ **then**
12: 将 v 加入社区 com_i;
13: **end if**
14: **end for**
15: **while** com_i 成员有增加
16: 根据现有社区大小更新负载 load;
17: $i \leftarrow i+1$; //社区编号自加
18: **end for**
19: 根据式 (3) 计算社区间通信量 C_{ij};
20: **return** $\{\mathrm{Com}_i\}$, $\{C_{ij}\}$.

有成员的计算总复杂度可近似为 $O(d\theta|V|)$. 计算 n 个社区两两之间通信量的时间复杂度 (第 19 行) 可近似为 $O(\frac{|V|(n^2-1)}{2n})$. 故算法 2 总时间复杂度为 $O(d|V|+n|V|+d\theta|V|+\frac{|V|(n^2-1)}{2n})$, 简化后可得 $O((d\theta+n)|V|)$. 若将 d 和 θ 视为常数, 则算法 2 所描述的社区划分算法的时间复杂度可近似表示为社区数 n 与用户数 $|V|$ 的乘积.

再分析 SDP 算法的时间复杂度, 服务器分组 (第 1 行) 的时间复杂度为 $O(1)$, 计算 $l_{\text{fattree-}i}$ or $l_{\text{fabric-}i}$, $i=1,2,3$ (第 2 行) 的时间复杂度为 $O(\frac{3|S|(|S|-1)}{2})$. 第 1 轮调用算法 2 (第 3 行), 针对 Fat-tree 和 Fabric 两种拓扑结构的时间复杂度分别为 $O(k|V|)$ 和 $O(p|V|)$, 第 1 轮划分的社区数分别为 k 和 p, 计算第 1 层通信量 $C_{\text{fattree-1}}$ 和 $C_{\text{fabric-1}}$ (第 4 行) 的时间复杂度分别为 $O(\frac{k(k-1)}{2})$ 和 $O(\frac{p(p-1)}{2})$, Hash 映射 (第 6 行) 复杂度分别为 k 和 p. 第 2 轮将第 1 轮划分出的 k 和 p 个社区, 每个再分别划分为 $k/2$ 和 k_a 个社区, 并与相应数目的服务器分组进行 Hash 映射. 假设第 1 轮划分社区的平均用户数分别为 $|V_{\text{fattree1}}|$ 和 $|V_{\text{fabric1}}|$, 则第 2 轮划分社区 (第 10 行) 的复杂度分别为 $O(\frac{k^2|V_{\text{fattree1}}|}{2})$ 和 $O(pk_a|V_{\text{fabric1}}|)$, Hash 映射 (第 12 行) 时间复杂度分别为 $k^2/2$ 和 pk_a, 计算第 2 层通信量 $C_{\text{fattree-2}}$ 和 $C_{\text{fabric-2}}$ (第 14 行) 的时间复杂度分别为 $O(\frac{k^2(k-2)}{4})$ 和 $O(\frac{pk_a(k_a-1)}{2})$. 同理, 可进一步推导出第 3 轮划分社区 (第 17 行) 的时间复杂度分别为 $O(|S||V_{\text{fattree2}}|)$ 和 $O(|S||V_{\text{fabric2}}|)$, 其中 $|V_{\text{fattree2}}|$ 和 $|V_{\text{fabric2}}|$ 分别表示两种拓扑结构下第 2 轮划分出社区的平均用户数, Hash 映射 (第 19 行) 的时间复杂度均为 $|S|$, 计算第 3 层通信量 $C_{\text{fattree-3}}$ 和 $C_{\text{fabric-3}}$ (第 21 行) 的时间复杂度分别为 $O(\frac{|S|(k-2)}{2})$ 和 $O(\frac{|S|(k-k_f-1)}{2})$. 在每一轮中社区的平均用户数可以用 $|V|\theta$ 除以该轮划分的社区数近似表示, 对于 Fat-tree, 有 $|S|=k^3/4$, 将所有执行步骤的时间复杂度求和化简后可得 SDP 算法的时间复杂度近似为 $O(|S|^2+k|V|)$. 对于 Fabric, 若将 p, k_a 和 k_f 均视为常数, 则 SDP 算法时间复杂度近似为 $O(|S|^2+|S||V|)$.

3.4 社交网络增量调整讨论

在求解社交网络数据放置问题过程中, 上述算法针对给定的社交网络及其用户交互情况求解出最佳数据放置方案. 然而社交网络现正处于快速发展期, 面对不断增长的社交用户量, 需要解决用户数据的增量放置问题. 针对这一问题, 本文提出一种用户数据的增量调整方法. 首先, 将社交网络的动态变化分解为以下 3 种情形分别加以讨论.

(1) 新用户加入网络. 当一个新用户 u 注册了一个社交账号, 他的数据可以随机放置在任何一台服务器上, 因为 u 还没有和其他用户建立关系, 在社交图上他是一个孤立的点. 从负载均衡的角度考虑, 将其放置在当前负载最轻的服务器上. 在实际中新用户 u 注册之后很可能立刻建立自己的社交圈, 即 $N(u)$, 例如添加好友或关注他人等, 从而快速融入到社交网络中. 针对这种情况, 首先将 u 到 $N(u)$ 内每个邻居 v 的读频度 r_{uv} 默认置为 1, 因为添加或关注操作需要读取对方的数据, 若除此以外用户 u 还有进一步的访问行为, 则 r_{uv} 按实际访问频度取值. 然后计算出 u 到所有邻居所在服务器的读通信量, 即 $\sum_{v\in N(u)\cap\phi_x} r_{uv}$, 选择通信量最大的服务器存放 u, 即 $\arg\max_x \sum_{v\in N(u)\cap\phi_x} r_{uv}$.

(2) 用户间交互频度发生变化. 以用户 u 和 v 为例, 当用户 u 到 v 的访问频度 r_{uv} 发生变化, 此时若他们已经存放在同一台服务器, 则不需要调整. 否则, 计算 u 所在服务器 x 上所有访问 v 的频度之和, 即 $\sum_{k\in N(v)\cap\phi_x\wedge u\in\phi_x} r_{kv}$, 若其值大于 w_v, 则在服务器 x 上创建 v 的副本.

(3) 用户自身更新频度发生变化. 当用户 u 的更新频度 w_u 发生变化, 找出所有满足$\sum_{k\in N(u)\cap\phi_x} r_{ku} > w_u$ 的服务器 x, 若 x 上存放 u 的副本且不是主副本, 则将其删除. 同样找出所有满足 $\sum_{k\in N(u)\cap\phi_x} r_{ku} < w_u$ 的服务器 x, 若 x 上未存放 u 的副本, 则创建一个.

随着新用户的加入, 服务器上存储的用户数据越来越多, 当用户规模增长到一定程度时, 现有的服务器规模将无法承载, 需要添加新的服务器以扩展存储系统. 所以, 在上述讨论的调整方法中, 当满足创建副本条件时, 还需再判断现有服务器存储容量是否已饱和, 若是, 则需要扩展系统. 扩展方式可以简单地分为 3 种. 第一, 增加交换机和服务器, pod 内服务器数量不变, pod 数量增加; 第二, 添加服务器, 升级现有交换机, 使其接口数量增加, 从而使得每个 pod 能够容纳更多的服务器, pod 数量保持不变; 第三, pod 数量和 pod 内服务器数量都增加, Fat-tree 只能采用该方式扩展, 因为其交换机接口和 pod 数同为 k. 不管哪种方式, 一旦系统规模扩展, 数据放置方案需要重新计算. 不同之处在于, 对于第 2 种方式, 同一 pod 内有新旧服务器, 新服务可以存储超出旧服务器存储能力的用户数据, 这样避免了对整个社交网络重新划分, 只需要分别对同一 pod 内或同一接入/access 层交换机的社交网络用户执行划分和映射; 而其余两种方式, 新旧服务器分布在多个 pod 内, 需要对整个社交网络从核心/Spine 层开始重新执行 SDP 算法, 逐轮划分和映射.

3.5 算法分布式实现讨论

本文所提出的 SDP 算法是一种集中式的算法, 算法的单机执行方式受限于运行服务器的配置, 在实际应用场景中, 社交网络规模巨大, 需要以分布式的方式执行. 本节将对 SDP 算法的分布式并行实现进行讨论.

SDP 算法利用多轮迭代执行重叠社区划分和社区–服务器 (分组) 映射来实现优化的数据放置, 因而分布式方式也需要通过多轮迭代完成. 除第 1 轮仅执行一次重叠社区划分外, SDP 算法其余两轮均多次调用 parOverlapComm 划分社区, 且同一轮内的调用相互独立, 故可采用分布式方式并行执行社区划分. 例如, 对于 Fat-tree, 第 1 轮将社交图 G 划分为 k 个社区并与 k 个 pod 一一对应; 第 2 轮将这 k 个社区并行再划分, 每个社区划分为 $k/2$ 个社区, 并与接入层交换机一一对应; 第 3 轮同样将上

一轮得到的社区以并行方式每个再划分为 $k/2$ 个社区, 并与服务器一一映射.

parOverlapComm 算法能够划分的社交网络规模同样受限于单台服务器的配置, 例如, 当网络用户规模达到千万级别时, 处理数据量接近 500 GB [25], 超出普通服务器的内存容量. 虽然 SDP 算法在第 2, 3 轮处理的社交图只是整个社交网络的一部分, 但是第 1 轮需要对整个社交网络进行划分, 因此, 同样需要讨论 parOverlapComm 算法的分布式并行实现.

根据主流的分布式实现框架, 如 Spark, 分布式环境主要由 1 个 master 服务器和 m 个 worker 服务器构成. 首先对社交网络数据进行预处理, 将社交图随机分割为 m 个规模相近的连通子图, 并分别存储在 m 个 worker 上. 除此以外, 为了便于后续计算, 每个连通子图的邻居集及其社交关系边也在对应的 worker 上以副本形式存储. 算法分两轮执行, 第 1 轮找出活跃度最高的 top-n 用户, 具体步骤如下:

(1) 每个 worker 计算对应的连通子图中所有用户的活跃度, 并找出 top-n 用户;

(2) 将所有 worker 找出的 top-n 用户聚合排序, 得出整个网络的 top-n 用户以及他们所在的 worker.

第 2 轮中每个 worker 均执行重叠社区划分, 具体步骤如下:

(1) 对于所有的 worker, 若 worker 不包含 top-n 用户, 则等待直到有消息到达转 (3), 否则转 (2);

(2) 对包含 1 个及以上 top-n 用户的 worker, 每个 worker 执行 parOverlapComm 算法第 4∼18 行, 其中 (i) 为该 worker 上的每个 top-n 用户初始化一个社区并编号, 执行 parOverlapComm 算法第 1 行为该社区初始化负载 load; (ii) 执行到第 7 行时, 若社区邻居集中有部分邻居不属于该 worker 对应的连通子图, 由于在数据预处理阶段这些邻居的副本已存储在该 worker 上, 故仍可执行后面的计算和判断; (iii) 执行到第 12 行时, 用户 v 加入社区 com_i, 若 v 有部分或全部邻居不在该 worker, 则将 v 及所属社区 com_i 以消息形式发送给所有包含其邻居的 worker; (iv) 执行到第 16 行时, worker 根据各社区负载计算平均负载, 并广播给其他 worker, 每个 worker 根据收到的值再计算平均负载;

(3) 当一个 worker 收到来自其他 worker 的消息时, 从消息中提取 v 及所属社区 com_i, 以 v 为初始点执行 parOverlapComm 算法第 6∼18 行;

(4) 当所有 worker 上的社区不再发生改变, 表示社区划分已完成. 每个 worker 根据式 (3) 计算其上的社区间通信量, 聚合所有 worker 上的计算结果得出所有社区及其相互间通信量.

4 算法实验

本节通过仿真实验对数据放置算法的有效性进行验证. 首先介绍实验设置, 然后通过多组实验分析不同参数下本文算法与对比算法的性能状况.

4.1 实验设置

实验基于对 Facebook 采集的真实数据集 [25] 进行仿真, 该数据集包含超过 1 千万用户, 从中选择 7 个区域子集作为本实验输入数据集, 它采集了包括美国 San Francisco, New York 等地区 461.3 万用户和 1.02 亿个社交关系边. 由于数据集中未包含诸如用户页面浏览的读交互信息, 而读操作在交互中占相当大的比重, 根据文献 [26] 揭示的规律生成用户间读操作, 用户读操作频度符合 Zipf 分布 ($\alpha = 0.72,\ \beta = 697$). 负载均衡阈值 $\mathrm{gini}^* = 0.2$, 系统冗余度阈值 $\theta^* = 5$. 分布式 SDP 算法在 Spark 平台上实现, 硬件环境包括 1 台 master 服务器, 12 台 worker 服务器, 每台服务器 CPU 为 Intel(R) Xeon(R) E5-2620 v2 @ 2.10 GHz × 2, 120 GB 内存.

表 2 拓扑结构参数设置

Table 2 Topological structure parameter settings

Fat-tree		Fabric					
k	n	k	k_a	k_f	k_s	p	n
8	128	8	8	4	2	4	128

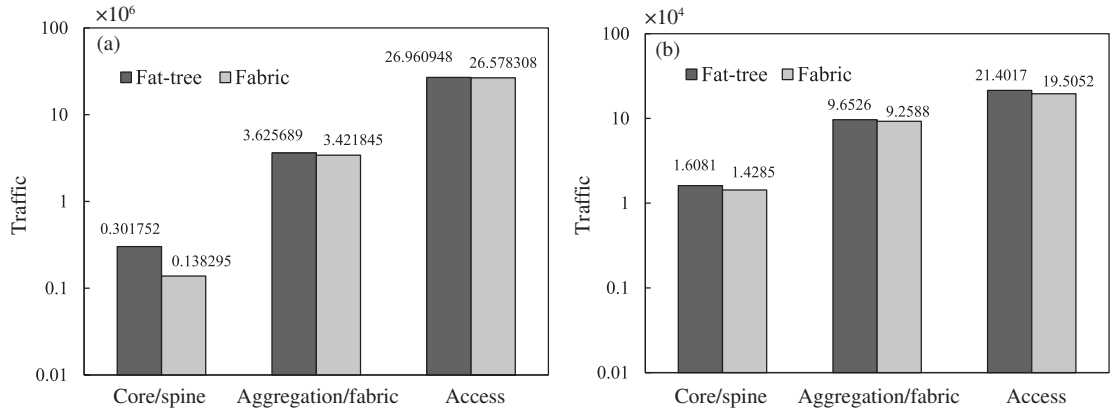

图 6 两种拓扑结构下各层通信量比较

Figure 6 Traffic comparison among different layers under two topologies. (a) Traffic distribution among different layers; (b) average traffic for each layer

4.2 实验结果及分析

本文实验共分 4 组, 分别从数据中心拓扑结构、数据中心网络规模、负载均衡约束和社交网络增量变化 4 个方面进行实验分析, 并将已有算法与本文所提 SDP 算法进行比较, 充分考察了各种算法所对应的通信量在两种拓扑结构下随着网络规模和负载均衡约束的变化情况, 从而验证 SDP 算法的优势.

4.2.1 两种拓扑结构的比较

为了便于比较, 本文为 Fat-tree 和 Fabric 配置相同数目的服务器 ($n = 128$), Fat-tree 的所有交换机和 Fabric 的接入交换机均为 8–口交换机, 其他参数设置如表 2 所示, 两种结构在各层的交换机数分别为 (16, 32, 32) 和 (8, 16, 32), Fat-tree 包含 8 个 pod, Fabric 包含 4 个 pod. 我们运行分布式 SDP 算法来优化数据放置, 两种拓扑结构下运行时间分别为 236 和 248 s, 各层通信量分布如图 6(a) 所示, 通信量由底向上呈递减分布, 主要集中在接入层, 约占总通信量的 87.6%, 而最顶层的通信量所占比例不到 1%, 说明本文算法 SDP 能够实现位置局部性, 将绝大部分通信限制在底层完成. Fat-tree 在各层的通信量均高于 Fabric, 尤其在最顶层, Fabric 通信量仅为 Fat-tree 的 45.8%, 这是因为 SDP 算法在第 1 轮中为 Fat-tree 划分 8 个社区, 而 Fabric 划分 4 个社区, 社区数即 pod 数, 社区数越少则社区间通信量越低. 此外, 还对经过每层交换机的平均通信量进行了统计, 从图 6(b) 可见, Fabric 略优于 Fat-tree.

一个 pod 所能容纳的服务器数目受限于交换机接口数目, Fat-tree 为了实现无阻塞通信让汇聚层与接入层拥有相同数量的交换机, 且按照完全二分图互连, 若增加过多交换机, 则交换机间互连势必会

表 3　不同规模拓扑参数设置

Table 3　Topological structure parameter settings with different scales

Fat-tree		Fabric					
k	n	k	k_a	k_f	k_s	p	n
12	432	12	18	4	4	3	432
20	2000	20	25	4	4	5	2000

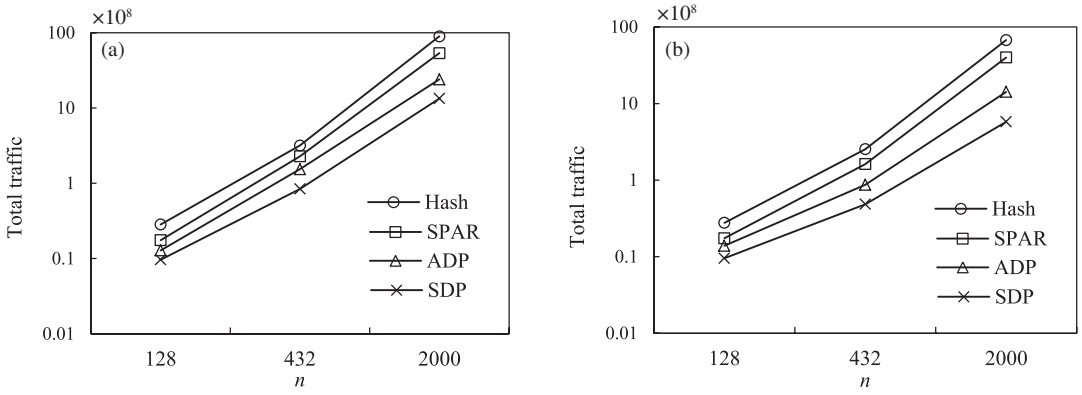

图 7　不同规模拓扑下放置算法比较

Figure 7　The comparison of algorithms under varied sizes of network topologies. (a) Fat-tree; (b) Fabric

占用接入服务器的接口, 从而限制 pod 内的服务器数目. Fabric 在汇聚层之上利用光纤交换机互连实现无阻塞通信, pod 内仅用少量 fabric 交换机便可互连大量接入交换机, 从而扩充了容纳的服务器数量. 因此, Fabric 较 Fat-tree 具有更好的扩展性能.

4.2.2　不同数据中心网络规模的比较

本组实验对比考察数据放置算法在不同规模 ($n = 128,\ 432,\ 2000$) 网络下通信量的变化, 此处通信量为各层通信量之和, 网络拓扑参数配置由表 3 给出. 作为对比, 本文分别实现了 Hash, SPAR [6] 和 ADP [11] 3 种算法, 由于这 3 种算法均未考虑数据中心网络拓扑结构, 为了便于比较, 增加划分结果到服务器的映射, 将 3 种算法带入本文算法 1 替换 parOverlapComm 算法. 需要指出的是这 3 种算法除 Hash 外本身并没有分布式实现, 故将社交网络规模缩小为 179.1 万用户和 4.08 千万社交关系边, 以使得单个服务器可以处理. 图 7 给出通信量比较结果, 随着服务器数目的增加, 将会有更多的社区被划分出来与服务器匹配, 服务器间的通信量也随之有很大增长. 由于 Hash 未对放置做任何优化, 其产生的通信量最高, SPAR 通过为不在同一服务器的邻居用户创建副本降低跨服务器读通信量, 但副本数目随着网络规模迅速增长, 导致副本更新通信量急剧增加. ADP 利用超图分割算法 PaToH [27] 对社交图进行社区划分, 再根据社区放置的服务器位置确定副本放置方案, 而本文算法 SDP 在划分社区的同时创建副本, 从实验结果可以看出同步优化的方法比分开异步优化的效果更好.

各算法的执行时间如表 4 所示, Hash 算法执行速度最快, 因为其主要操作是执行 Hash 函数和负载均衡判定. SPAR 算法在每一轮通过添加、删除用户节点和边来确定用户所属分区及副本数目, 这一过程较为耗时, 故执行时间较长. ADP 算法每一轮都执行超图分割和 Hash 映射, 再计算副本数量和位置, 执行时间在所有算法中最长. SDP 算法中每个用户节点都需要多次计算加入社区对通信量的

表 4 不同规模拓扑下算法运行时间

Table 4 The running time of algorithms under varied sizes of network topologies

n	Hash		SPAR		ADP		SDP	
	Fat-tree (s)	Fabric (s)	Fat-tree (min)	Fabric (min)	Fat-tree (min)	Fabric (min)	Fat-tree (s)	Fabric (s)
128	23	24	32	47	66	79	207	218
432	21	25	41	58	103	136	253	277
2000	20	23	128	175	297	378	725	776

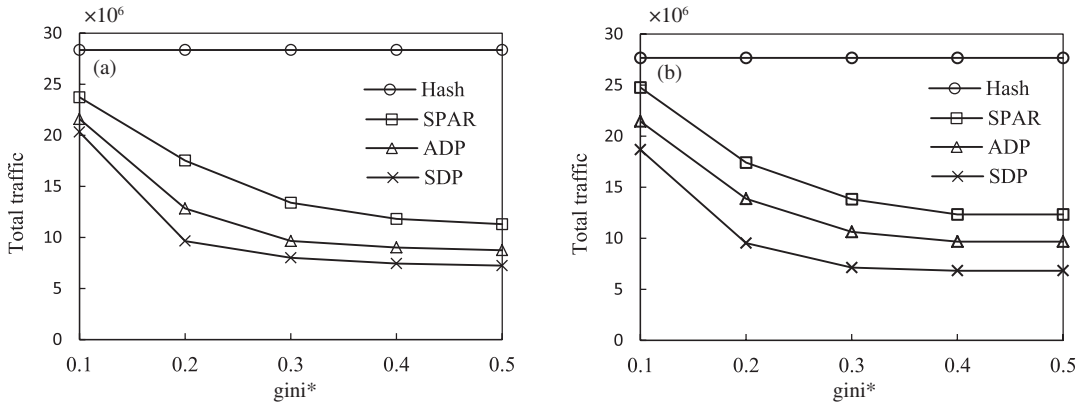

图 8 不同负载均衡约束下的放置算法比较

Figure 8 The comparison of algorithms under varied load balancing constraints. (a) Fat-tree; (b) Fabric

影响, 但这一过程可以并行执行, 故执行时间不会太长, 明显低于 SPAR 和 ADP 两种算法. 各算法在 Fabric 拓扑下的执行时间均不同程度高于 Fat-tree, 这与网络拓扑的具体结有关. 在本实验中相同规模下前者的 pod 数和接入层交换机数均小于后者, 在算法 1 的分布式实现中前者的并行度更低. 除 Hash 外其他算法的执行时间随着数据中心网络规模的增长而变长, 因为问题规模随之增大, 数据副本放置的位置有更多选择, 而 Hash 操作执行时间与服务器数量无关, 随着服务器增加, 并行度呈增长趋势, 故执行时间有所下降.

4.2.3 不同负载均衡的比较

要达到良好的可扩展性, 除了降低通信量之外, 还需要均衡各服务器的负载. 本文用 Gini 系数衡量负载均衡水平, 数值越低均衡水平越高, 对数据放置的约束也就越大. 本组实验对比考察放置算法在不同均衡约束下的通信量变化, 社交网络规模与 4.2.2 小节保持一致, 数据中心网络规模 $n = 128$, 结果如图 8 所示. 当负载均衡阈值 gini* 增加时, 除 Hash 算法外其他放置算法的通信量均有所下降, 因为 Hash 算法能够让数据在服务器间的分布十分均匀, 且低于阈值, 故不受影响. 通信量并不会随着 gini* 增加一直持续降低, 而是降幅逐渐趋缓, 因为 gini* 越大, 对数据划分的约束越小. 当 gini$^* = 0.5$ 时, 所有算法均不再受到影响. 本文 SDP 算法采用种子扩张的方法划分社区, 且允许社区重叠, 在一定程度上有助于平衡各社区规模. 当 gini$^* \geqslant 0.2$ 时, 通信量仅略有下降, 故在其他组实验中的 gini* 值默认设为 0.2, 既保证负载均衡水平, 又可获得较低的通信量.

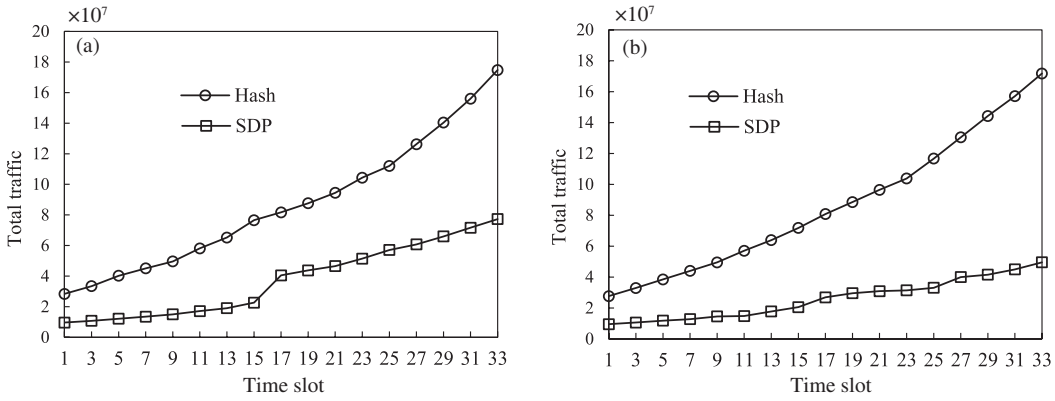

图 9　面向社交网络规模增长的增量调整

Figure 9　Incremental adjustment for social network scale growth. (a) Fat-tree; (b) Fabric

4.2.4　社交网络增量变化

社交网络规模处于不断增长状态, 本组实验考察在增长过程中数据放置增量调整及其产生的通信量. 为了模拟社交网络增长, 本文采用离散时隙, 初始规模为 179.1 万用户和 4.08 千万社交关系边, 每经过一个时隙, 用户量增长 3%, 社交关系边随之增长. 数据中心网络规模为 $n = 128$, 每台服务器存储容量为 5.5 万用户数据. 实验从第 1 时隙开始直到第 33 时隙为止, 社交网络规模增长到 461.3 万用户, 增长近 2.6 倍. 本文比较了 Hash 与 SDP 算法, 另外两种算法由于社交网络规模增长超过单机处理能力, 未作比较. 实验结果如图 9 所示, 通信量随着用户增加逐步增大, 在第 17 时隙时 SDP 算法在两种拓扑结构下的增幅均有所提高, 这是因为此时用户量达到 295.8 万, 存储的用户副本数已超过服务器总容量, 故需要扩展服务器规模, Fat-tree 由 $n = 128$, $k = 8$ 扩展到 $n = 250$, $k = 10$, 而 Fabric 由 $n = 128$, $k = 8$ 扩展到 $n = 192$, $k = 10$. 扩展后重新执行数据放置, Fat-tree 结构较 Fabric 拥有更多服务器, 所以其通信量增加更多. Hash 算法由于不创建副本, 所以没有发生超出存储容量而扩展服务器规模的情况, 通信量一直稳步增长. 当到达第 27 时隙时, 数据量再次超过 Fabric 结构下服务器总容量, 继续扩展服务器规模至 $n = 256$, $k = 12$, 重新执行一次数据放置的全局优化, 故从图 9(b) 可以看出通信量在第 27 时隙增幅稍有增大.

5　结束语

数据中心内频繁的跨服务器和交换机通信已经成为制约在线社交网络扩展的关键因素. 本文正是从支持扩展的角度对具有相关性的用户数据放置问题展开研究. 从优化社交局部性和位置局部性两个方面入手, 一方面, 将数据放置问题转化为带重叠的社区划分问题, 通过重叠用户的发现, 自然引入复制方法, 不同于以往方法的是我们让复制与划分同步进行, 充分发挥二者在最小化通信量方面的优势; 另一方面, 在社区划分的基础上, 进一步将问题转化为社区–服务器二次分配问题, 将彼此间通信量较高的社区映射到距离临近的服务器. 基于 Facebook 数据集上的比较实验结果验证了本文所提 SDP 算法的效率和优势.

参考文献

1　Shvachko K, Kuang H, Radia R, et al. The hadoop distributed file system. In: Proceedings of the 26th Symposium

on Mass Storage Systems and Technologies, Nevada, 2010. 1–10
2 DeCandia G, Hastorun D, Jampani M, et al. Dynamo: amazon's highly available key-value store. In: Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles, Washington, 2007. 205–220
3 Lakshman A, Malik P. Cassandra: a decentralized structured storage system. Operat Syst Rev, 2010, 44: 35–40
4 Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput, 1998, 20: 359–392
5 Chen H H, Jin H, Wu S L. Minimizing inter-server communications by exploiting self-similarity in online social networks. IEEE Trans Parall Distr Syst, 2016, 27: 1116–1130
6 Pujol J M, Erramilli V, Siganos G, et al. The little engine(s) that could: scaling online social networks. IEEE/ACM Trans Netw, 2012, 20: 1162–1175
7 Tran D A, Nguyen K, Pham C. S-clone: socially-aware data replication for social networks. Comput Netw, 2012, 56: 2001–2013
8 Liu G X, Shen H Y, Chandler H. Selective data replication for online social networks with distributed datacenters. IEEE Trans Parall Distr Syst, 2016, 27: 2377–2393
9 Jiao L, Li J, Du W, et al. Multi-objective data placement for multi-cloud socially aware services. In: Proceedings of IEEE Conference on Computer Communications, Toronto, 2014. 28–36
10 Tran D A, Zhang T. S-put: an ea-based framework for socially aware data partitioning. Comput Netw, 2014, 504–518
11 Yu B Y, Pan J P. Location-aware associated data placement for geo-distributed data-intensive applications. In: Proceedings of IEEE Conference on Computer Communications, Hong Kong, 2015. 603–611
12 Yu B Y, Pan J P. Sketch-based data placement among geo-distributed datacenters for cloud storages. In: Proceedings of the 35th Annual IEEE International Conference on Computer Communications, San Francisco, 2016. 1–9
13 Tang J, Tang X Y, Yuan J S. Optimizing inter-server communication for online social networks. In: Proceedings of the 35th International Conference on Distributed Computing Systems, Columbus, 2015. 215–224
14 Jiao L, Li J, Xu T Y, et al. Optimizing cost for online social networks on geo-distributed clouds. IEEE/ACM Trans Netw, 2016, 24: 99–112
15 Gregory S. Finding overlapping communities in networks by label propagation. New J Phys, 2010, 12: 103018
16 Lancichinetti A, Fortunato S, Kertesz J. Detecting the overlapping and hierarchical community structure in complex networks. New J Phys, 2009, 11: 1–18
17 Lee C, Reid F, McDaid A, et al. Detecting highly overlapping community structure by greedy clique expansion. 2010. arXiv:1002.1827
18 Han Z M, Tan X S, Chen Y, et al. NCSS: an effective and efficient complex network community detection algorithm. Sci Sin Inform, 2016, 46: 431–444 [韩忠明, 谭旭升, 陈炎, 等. NCSS: 一种快速有效的复杂网络社团划分算法. 中国科学: 信息科学, 2016, 46: 431–444]
19 Qiao S J, Han N, Zhang K F, et al. Algorithm for detecting overlapping communities from complex network big data. J Softw, 2017, 28: 631–647 [乔少杰, 韩楠, 张凯峰, 等. 复杂网络大数据中重叠社区检测算法. 软件学报, 2017, 28: 631–647]
20 Benevenuto F, Rodrigues T, Cha M, et al. Characterizing user behavior in online social networks. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, Chicago, 2009. 49–62
21 Li D, Chen G H, Ren F Y, et al. Data center network research progress and trends. Chinese J Comput, 2014, 37: 259–274 [李丹, 陈贵海, 任丰原, 等. 数据中心网络的研究进展与趋势. 计算机学报, 2014, 37: 259–274]
22 Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. In: Proceedings of ACM SIGCOMM Conference on Data Communication, Seattle, 2008. 63–74
23 Clos C. A study of non-blocking switching networks. Bell Labs Tech J, 1953, 32: 406–424
24 Roy A, Zeng H Y, Bagga J, et al. Inside the social network's (datacenter) network. In: Proceedings of ACM SIGCOMM Conference on Data Communication, London, 2015. 123–137
25 Wilson C, Sala A, Puttaswamy K P N, et al. Beyond social graphs: user interactions in online social networks and their implications. ACM Trans Web, 2012, 6: 1–31
26 Jiang J, Wilson C, Wang X, et al. Understanding latent interactions in online social networks. ACM Trans Web, 2013, 7: 1–39
27 Catalyurek U V, Aykanat C. PaToH (partitioning tool for hypergraphs). In: Encyclopedia of Parallel Computing.

Berlin: Springer, 2011. 1479–1487

Data placement approach for scalable online social networks

Jingya ZHOU[1,2,3*], Jianxi FAN[1] & Jin WANG[1]

1. *School of Computer Science and Technology, Soochow University, Suzhou* 215006*, China*;
2. *Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing* 210093*, China*;
3. *Jiangsu Provincial Key Laboratory of Computer Information Processing Technology, Suzhou* 215006*, China*
* Corresponding author. E-mail: jy_zhou@suda.edu.cn

Abstract Online social networks are attracting more and more users. Faced with hundreds of millions of users, how to store user data in a scalable manner has become a hot issue of focus for both social service providers and researchers. Currently, distributed key value store is widely used; it places user data across multiple storage servers based on a hash approach. However, it results in a huge amount of communication traffic inside a data center, and is not conducive to the scale of social networks. By considering user interaction characteristics, this paper proposes a data placement approach that combines both social graph partitioning and data replication. Considering the network topologies of data centers, we design the data placement for specific topologies. Furthermore, we discuss the incremental adjustment for social network growth and the distributed implementation of the proposed algorithms. Finally, experiments on real world traces indicate that the proposed algorithms can effectively reduce internal communication traffic, thereby enhancing the scalability of online social networks.

Keywords online social networks, data placement, data center networks, social locality, position locality

Jingya ZHOU was born in 1983. He received B.S. and Ph.D. degrees in computer science from Anhui Normal University and Southeast University, China, in 2005 and 2013. He is currently an assistant professor with the School of Computer Science and Technology, Soochow University, China. His research interests include cloud computing, edge and fog computing, parallel and distributed systems, online social networks, and data center networking.

Jianxi FAN was born in 1965. He received the B.S., M.S., and Ph.D. degrees in computer science from Shandong Normal University, Shandong University, and City University of Hong Kong, China, in 1988, 1991 and 2006, respectively. He is currently a professor with the School of Computer Science and Technology, Soochow University, China. His research interests include parallel and distributed systems, interconnection architectures, the design and analysis of algorithms, and graph theory.

Jin WANG was born in 1985. He received a B.S. degree from the Ocean University of China in 2006, and a Ph.D. degree in computer science jointly awarded by City University of Hong Kong and the University of Science and Technology of China in 2011. He is an associate professor with the School of Computer Science and Technology, Soochow University, China. His research interests include network coding, information-centric networks, and Fiber-wireless networking.