P2P网络中基于节点负载度的均衡控制算法研究

陈立龙[1] 刘玉华[1] 许凯华[2] 魏玉英[3]
(华中师范大学计算机科学系 武汉 430079)[1] (华中师范大学物理学院 武汉 430079)[2]
(华中师范大学教育部教育信息技术工程研究中心 武汉 430079)[3]

摘 要 在非结构化P2P网络中,为了抑制"搭便车"行为,需要寻找网络中的集散节点。在参考传统集散节点连接数的基础上,还考虑了不同节点的负载能力差异,引入了负载度概念,将负载度高的节点称为重载节点。在此基础上提出了一种节点负载均衡控制算法,将重载节点的负载向一些轻载节点转移。仿真实验表明,本算法能有效地控制网络中各节点的负载,使之尽可能合理均衡分布,以此抑制"搭便车"行为,从而维护和提高网络性能。

关键词 P2P网络,搭便车行为,负载度,备用节点表,均衡控制算法

Balance Control Algorithm Based on Node Load Degree over Peer-to-Peer Network

CHEN Li-long[1] LIU Yu-hua[1] XU Kai-hua[2] WEI Yu-ying[3]
(Department of Computer Sciences, Huazhong Normal University, Wuhan 430079, China)[1]
(College of Physics, Huazhong Normal University, Wuhan 430079, China)[2]
(Engineering & Research Center for Information Technology on Eduction, Huazhong Normal University, Wuhan 430079, China)[3]

Abstract In unstructured peer-to-peer network, we need to find hubs to restrain "free-riding" behavior. In this paper, based on referring previous connection numbers from hubs, we took nodes' power difference into account, also introduced the concept about load degree and those nodes holding high load degree are called overload nodes. Then a load balance control algorithm was proposed, in which load from overload nodes is shifted to light-load nodes. Simulation results shows the algorithm can effectively balancing nodes' load and load distribution is more balanced, so the algorithm can effectively restrain "free-riding" behavior and helps to maintain and improve network performance.

Keywords Peer-to-Peer, Free-riding behavior, Load degree, Backup node table, Balance control

1 前言

近年来,P2P网络在文件共享、流媒体上的应用越来越受到人们的关注。然而,由于P2P网络中的节点行为特征具有较大的差异性[1],一些"搭便车"者也相继产生。人们把P2P网络中只享受信息资源服务而不为系统作贡献的节点称为"搭便"车者、"搭便车"行为或现象[1,2,6]。正是这些"搭便车"者或"搭便车"行为的出现,使得少量集散节点在有限的能力范围内却要维持大量的连接数[7],大量的"搭便车"节点不参与网络贡献。由此,倘若某些高连接数的集散节点失效或崩溃,则会使整个P2P网络性能大大降低,网络的稳定性和可靠性也得不到保障。所以近年来相继提出一些抑制"搭便车"行为的方法,概括起来主要有激励机制、博弈论以及社会网络或经济模型策略[2]。另外,也可以从网络拓扑结构出发,找出可能影响到网络结构的"集散节点",并采用相应的控制算法转移它们的负载[7]。但是在集散节点的评判标准上,以往的文献基本上都是以节点的连接数来进行说明的[3,4,7],只是所用名称不同而已。文献[4]中将P2P网络中连接数较高的节点称为"割点",而文献[3]则用"分点"这一结构性概念。由于P2P网络中节点间的能力相差极大,要想最后使整个P2P网络的负载达到"帕累托"最优[5],必须充分开发每个节点的潜能,使得能力值大的节点承担更多的网络负载。若笼统地以同一个连接数去衡量所有节点的能力,势必会忽视P2P网络中节点异构性的问题。

基于以上论述,本文根据节点能力大小,给出了负载度、重载节点、邻居负载均值等概念,提出了一种均衡负载的控制算法。算法通过节点负载度的不同,将网络中重载节点上的负荷向一些轻载节点转移。算法讨论了如何选择轻载节点及如何转移等具体问题。仿真实验表明,负载均衡算法较好地解决了非结构化P2P网络中集散节点的负载不均衡问题,以此可以抑制"搭便车"行为,从而维护和提高P2P网络性能。

2 基本概念与定义

以下给出本文在非结构化P2P网络环境中用到的一些概念与定义。

节点连接数定义:将节点 i 的入度与出度之和称为该节点的连接数,用 $L(i)$ 表示。其中入度与出度是根据节点间共享的文件数计算得来的。

到稿日期:2009-12-23 返修日期:2010-03-09 本文受国家自然科学基金项目(60673163)资助。
陈立龙(1985-),男,硕士生,主要研究方向为云计算、P2P网络和网络优化;刘玉华(1951-),女,博士,教授,博士生导师,主要研究方向为计算机网络与通信技术、无线网络、复杂网络等,E-mail:yhliu@mail.ccnu.edu.cn(通信作者)。

节点能力值定义：将节点 i 的计算能力、存储能力、带宽、在线时间等评价因素综合而形成的一个量化值称为该节点的能力值，用符号 $P(i)$ 表示。

节点负载度定义：将节点 i 的连接数与能力值的比值称为该节点的负载度，用 $D(i)$ 表示。

$$D(i)=L(i)/P(i)$$

邻居负载均值定义：将节点 i 在 t 时刻的邻居总负载度 sum 与邻居节点数 n 的比值称为该节点的邻居负载均值，用 $Navg(t)_i$ 来表示。

$$Navg(t)_i=sum/n$$

重/轻载节点定义：在某一时刻 t，若节点 i 此时的负载度大于它的邻居负载均值，则称该节点为重载节点，否则为轻载节点。

3 节点能力值的度量

对节点能力值的度量，最关键的地方在于如何适当地选取评价节点的因素。文献[11]列举出了 6 个可能体现节点异构性的因素：节点本地查询能力、节点维持稳定性的能力、节点维持连通性的能力、节点的存储能力以及节点处理信息的能力。文献[12]则从 CPU 计算能力、主存大小、网络带宽和活动时间 4 个方面评估节点的能力值，并要求在计算过程中网络带宽所占的比重最大。文献[13]则直接将节点的能力值看成是有关节点网络带宽的函数。

在选取节点能力值的评价因素上，可能随着网络环境以及相关的应用领域的不同发生很大的变化。例如，对于处于低带宽网络环境下的节点，可能带宽是影响节点能力值的决定性因素；相反，若对于高带宽网络环境下的节点能力值则可能更多地与 CPU 计算能力或存储能力有关。但是，选取评价因素无论如何都得遵守一些基本的原则。

(1)通用性：所选取的评价因素应该适用于 P2P 网络中所有的节点。

(2)真实性：由所选取的评价因素得出的能力值能真实地反映节点的能力差异。

4 节点负载均衡控制算法

在 P2P 网络中，节点负载分布失衡直接关系到网络的可靠性和稳定性。因此，如何有效地调节网络中各节点的负载度，就显得尤为重要。本文提出的负载均衡控制算法从网络的逻辑结构出发，为重载节点中的每个共享文件从网络中找到备用节点，并将重载节点与备用节点组成二叉树结构，如图 1 所示，然后将转向重载节点的部分请求迁移到备用节点，以达到整体均衡的目的。

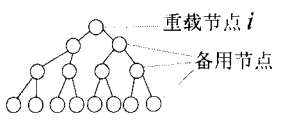

图 1 源节点与备用节点形成的二叉树

4.1 控制算法思想

在某一时刻，节点 i 处于重载状态，此时若节点 i 收到其它节点向它发出的文件 $file$ 请求，它会首先搜索自身存储的备用节点表。若能找到含有 $file$ 的可用备用节点(见图 1)，则直接将这一连接请求向备用节点转移。假如重载节点之前并未建立关于 $file$ 的备用节点表，就从节点 i 出发，搜索与它相连并含有 $file$ 文件的节点。最后在所有返回的节点中挑选出负载度最小的两个节点，作为节点 i 中关于 $file$ 文件的备用节点。若出现节点 i 无法找到备用节点或者找出的备用节点也均是重载节点而无法转移请求负载的情况，这时从节点 i 已找到的备用节点起再寻找与备用节点相连并含有 $file$ 文件的节点，依次进行下去，直到找到可供转移的轻载节点。若最后在 TTL 之内仍然未找到，就返回查找失败信息。

在具体的算法实现过程中，节点 i 所确立的关于资源 $file$ 的备用节点 l,r 的存储采取一种三元组结构($file,l,r$)，同时每个可能的重载节点各自维护着一张备用节点表 $List\_children$，该表记录着每个节点所确立的三元组集合。每个备用节点维护着一张父节点表 $List\_parent$，记录关于资源 $file$ 的父节点为 V，如表 1，表 2 所列。

表 1 List_children

资源	左子女	右子女
file	IP(l)	IP(r)

表 2 List_parent

资源	父节点
file	IP(V)

另外，节点间的连接信息是通过它们的邻接矩阵来实现的，只要节点 i,j 间有文件交换，则两节点的邻接矩阵分别增加 1。假如节点 i 仅仅分别与节点 j,k,l 存在连接，并且相互交换的文件数分别为 1,5,10，则节点 i 的邻接矩阵如图 2 所示，从图中可以看到，节点 i 仅仅知道自己的节点连接信息，对 j,k,l 之间的交换信息是未知的，因此将相应的数据置为 0。

$$\begin{bmatrix} 0 & 1 & 5 & 10 \\ 1 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 \end{bmatrix}$$

图 2 节点 i 的邻接矩阵

4.2 控制算法描述

均衡控制算法主要由以下 3 个步骤组成：

Step 1 节点 i 收到来自其它节点的文件连接请求，并判断节点自身是否为重载节点。

Step 2 若节点 i 为轻载节点，则直接响应 Step1 中的文件请求；若节点 i 为重载节点，此时节点 i 先搜索关于请求文件的备用节点表。

Step 3 若备用节点表中存在处于轻载状态的备用节点，则节点 i 直接将连接请求向备用节点转移；若备用节点表中不存在文件对应的备用节点或备用节点均处于重载状态，则需要重新建立关于此请求文件的二叉树备用节点表。

具体来讲，算法主要由以下两部分组成：重载节点的判断和建立二叉树备用节点表。控制算法中需要用到的参数符号如表 3 所列。

表 3 控制算法中参数列表

参数符号	参数说明
L(i)	节点 i 的连接数
P(i)	节点 i 的能力值
D(i)	节点 i 的负载度
Navg(t)	t 时刻邻居负载均值
start	起始节点(发出搜索请求节点)的编号
end	结束节点(接收搜索请求的节点)的编号
file	起始节点需要搜索查找的文件标识符
l	指向备用节点左子树的指针

r	指向备用节点右子树的指针
TTL	搜索跳数限制

4.2.1 确定重载节点算法

重载节点的查找主要依据节点的负载度与当前邻居全局负载度均值之间的比较，将那些比均值大的一些节点称为重载节点。确定重载节点详细算法见算法 1。

算法 1 重载节点确定算法

SearchOverloadNode(*end*, *Navg*(*t*)){

//输入：结束节点的编号以及此时刻邻居负载均值

//输出：重载节点的编号

Step 1 根据 *end* 节点的邻接矩阵统计与它相连的邻居结点数 n。

Step 2 利用 Gossip 协议[10]收集各邻居节点的负载度并统计邻居总负载度 *sum*。

Step 3 利用公式 $Navg(t)_i = sum/n$ 计算出此时的 *end* 节点的邻居负载均值。

Step 4 利用 $D(i)=L(i)/P(i)$ 计算出 *end* 节点的负载度 $D(end)$，并与 Step 1 计算出的 $Navg(t)_{end}$ 作比较。若前者大，则该节点为重载节点，否则为轻载节点。

Step 5 返回节点的编号值 *end*。

}

4.2.2 建立二叉树备用节点表算法

节点接收到文件连接请求后，若自身已经成为重载节点而不能直接响应请求，这时查找该文件对应的备用节点表，将连接请求向轻载备用节点转移；但是若备用节点表中的节点均是重载节点或者之前该节点并没有为该文件资源确立备用节点表，则需要从该节点出发向它所连接的节点发出搜索请求并寻找轻载节点。

算法 2 寻找备用节点算法

SearchBackupNode(*file*, *srart*, *end*){

//输入：起始节点需要搜索查找的文件标识符 *file* 以及结束节点的编号 *end*；

//输出：备用节点左右子树的指针（$*l$ 和 $*r$）

Step 1 根据节点 *end* 的邻接矩阵，找到与编号 *end* 节点相连的节点编号序列（根据邻接矩阵可以看出）。

Step 2 从 Step 1 所找出的节点中筛选出共享有文件标识符为 *file* 的节点列表。

Step 3 若返回 0 个节点，返回失败信息；若只返回 1 个节点，将其作为 *end* 节点的左子树；若返回 2 个节点，将负载度低的作为左子树，高的作为右子树；若返回多个节点（超过 2 个），就转向 Step 4。

Step 4 将由 Step 3 找出的节点按照负载度进行升序排列，并将前两个节点分别作为 *end* 节点的备用节点的左右子树。

Step 5 若左子树为轻载节点，返回左子树指针，否则从左右子树出发，重复执行前 5 步。

Step 6 若 *TTL* 未减少至 0，返回左右子树指针；否则，返回失败信息。

}

4.3 算法时间复杂度分析

在上节的控制算法中，最复杂的情况出现在：当节点 i 向节点 j 请求文件 *file* 时，因为节点 j 自身是重载节点而不能直接响应搜索请求，此时需要查找备用节点表，但其中并没有确立关于 *file* 的备用节点表，由此执行算法 2，寻找备用节点。设节点 j 所共享的文件数为 m，与节点 j 相连的节点数为 c（$c \leqslant N-1$）（N 为网络中总节点数）。首先需要搜索节点 j 的 *List_children* 表，表中至多有 m 项，则查找算法最坏情况下需要比较 m 次。其次在算法 2 中需要查找与节点 j 相连并共享有文件 *file* 的节点，按负载度的大小升序排列，与节点 j 相连的节点数最多为 $N-1$，则最坏情况下需要比较 $N-1$ 次。至于其中的排序问题，因为返回的节点的最大值为 $N-1$，考虑到实际网络中 N 值较大，采用快速排序算法最坏情况下需要比较 $N\log N$ 次。另外若节点 j 的备用节点依然是重载节点，则从备用节点出发继续向下搜索，直到 *TTL* 减少至 0，即查找备用节点需要重复 *TTL* 次。所以，整个算法的时间复杂度为 $m+(N+N\log N-1)*TTL$，其中 m，*TTL* 是可数的，即算法的总时间复杂度为 $O(N\log N)$。

5 仿真实验

5.1 实验方法和参数

实验是基于模拟的，利用 BA 算法随机产生一张连通的无向图作为实验网络拓扑的基础[6]。整个实验采用的方法是在相同文件请求连接数的条件下，通过对比有无控制算法分别返回的节点负载度分布图来表明均衡控制算法的有效性。与均衡控制算法实验有关的参数设置如表 4 所列。

表 4 模拟实验需要到的参数

参数符号	参数说明	设置值
N	网络节点总数	5000
FS	单个节点共享的文件数	100
FSS	网络总共享文件数	2000
P(i)	节点 i 的能力值	1～20
TTL	搜索跳数限制	2～5

5.2 实验过程与结果

这个实验的目的在于验证文中提出的负载均衡算法是否有效。对比实施控制之前的 P2P 网络，若根据返回的节点负载分布图可以看出实施控制算法后的节点负载分布更加均匀且比较密集，则表明算法是有效的。图 3、图 4 显示了两种状态在请求连接数为 50000 且负载度在 50 之内（节点负载度大概都落在这范围内）的节点负载分布情况。由此可以看到，未执行控制算法的 P2P 网络中节点的负载已经出现了极不平衡的现象。这是因为未采取控制算法的 P2P 网络中的节点连接数符合小世界网络[8,9]和幂律法则，大量的节点拥有极少数连接，而少数节点拥有极大的连接数，这样势必会导致节点负载的两极化。另外在实验过程中，随着文件请求连接数的增加，连接数高的节点获得多的连接数的几率也会很大，这样会加剧节点连接数的失衡而导致最后网络、节点负载的分散。而执行控制算法的 P2P 网络中节点负载分布十分均衡，这是因为虽然经过 BA 算法生成的网络是符合幂律法则的，但是实验过程中随着负载的转移，高负载节点的连接数并没有明显增加，而那些轻载节点的连接数逐渐变大，负载度也会相应增大，这样网络中节点的负载就会慢慢地达到一个平衡而逐渐趋于稳定。由此可见，均衡控制算法用于平衡网络中节点负载的效果是很明显的。

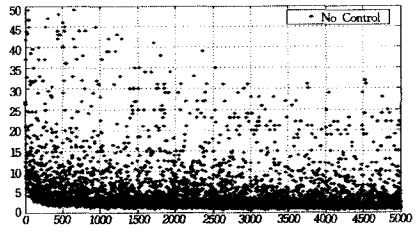

图 3 无控制算法下结点负载分布情况

（下转第 102 页）

ca2Dot 型传感器节点，每个节点每隔 31s(一个时间点)采集一个温度数据，传感器节点接受数据包的能耗为发送数据包能耗的 0.37 倍[6]。设置传感器保持 120 个最近时间点的数据，查询系统需要查询 K_i 个最大的数据，并假设在查询过程中 K_i 的固定值不变，评估在 Top-K 查询的过程中所有传感器节点传送和收到数据包的整体能耗。

传感器节点上传到父节点的数据包包括一个 32 比特的数据值部分和一个 32 比特的数据 ID 部分，其中数据 ID 部分包括一个 16 比特的节点 ID 和一个 16 比特的时间戳。Sink 节点回传到传感器节点的探寻包和更新滤波器包均为一个 32 比特的数据值(探寻条件值或滤波器值)。由于传感器节点收发数据包的能耗与数据包的大小成正比，为了简便起见，将发送一个 32 比特包定义为一个通信包，将收到一个 32 比特包折算为一个通信量的 0.37 倍，通过比较通信量可以评估算法的整体通信能耗。

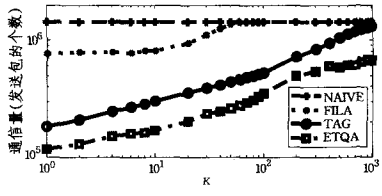

图 5 算法通信量仿真结果比较

为了与 ETQA 算法对比，实验同时仿真了 NAIVE、FILA 与 TAG 3 种算法作为对比。NAIVE 算法即无线传感器网络中所有节点将采集到的所有数据都上传到 Sink 节点的方法，简称 NAIVE 算法，横轴为 1 到 1000 之间选择的 28 个 K 值，纵轴为 2004 年 3 月 1 日一天数据的 Top-K 查询中所有节点的整体通信量，如图 5 所示。

通过通信量对比仿真发现，本文提出的 Top-K 查询算法 ETQA 比 NAIVE、TAG 和 FILA 算法能较大幅度地减少网络节点的整体通信量，从而大大地节省了传感器节点的整体能耗。由于 ETQA 算法中只对数据排序和比较，数据处理延时很小，并且算法中探寻和设置滤波器次数非常有限，同时通信数据量较小(如仿真中 31s 才采样一个数据)，通信延时也较小，因此整体上 ETQA 算法的时间复杂度与其他的 3 种算法相似。

结束语 为了节省通信量，本文提出一种结合具体应用的数据融合算法，以利用数据的相关性通过设置滤波器达到减少冗余数据的上传的目的。使用数据融合方法降低无线传感器节点整体通信量需要结合具体问题具体分析。

参 考 文 献

[1] Sakai H, Iiyamab S, Tokoc K. Evaluation of Water quality and pollution using multichannel sensors[J]. Sensors and Actuators B:Chemical, 2000, 66:1

[2] Silberstein A, Braynard R, Ellis C, et al. A sampling-based approach to optimizing top-k queries in sensor networks[C]// Proceeding of IEEE ICDE. 2006

[3] Madden S, Franklin M J, Hellerstein J M, et al. TAG: A Tiny Aggregation Service for Ad Hoc Sensor Networks[C]// Proc. Usenix Fifth Symp. Operating Systems Design and Implementation (OSDI '02). Dec. 2002:131-146

[4] Wu M, Xu J, Tang X, et al. Top-k monitoring in wireless sensor networks[J]. IEEE Trans. on Knowledge and Data Engineering, 2007, 19:7

[5] http://db.csail.mit.edu/labdata/labdata.html

[6] http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2DOT_Datasheet.pdf

(上接第 88 页)

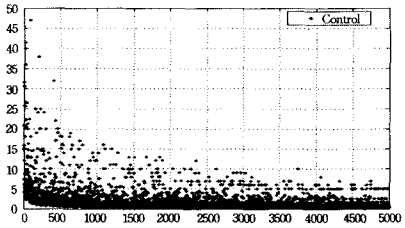

图 4 控制算法下结点负载分布情况

结束语 本文首先指出针对高异构性的 P2P 网络，连接数根本不能较准确地反映网络中的“集散节点”，所以我们采用节点的负载度来进行衡量，并将负载度高于邻居负载均值的节点称为重载节点。采用均衡控制算法将它的部分负载向轻载节点转移。相比未采取控制算法的 P2P 网络，节点负载度分布十分均衡。我们下一步的工作首先要考察节点能力值的时刻变化和高动态性情况下算法的运行情况，还要尝试把此均衡控制算法应用于结构化 P2P 网络。

参 考 文 献

[1] Adar E, Huberman B. Free riding on Gnutella[J]. First Monday, 2000, 5(10):32-35

[2] Ramaswamy L, Liu L. Free riding: A new challenge to Peer-to-Peer file sharing systems[C]// Proceedings of t he 36th Hawaii International Conference on System Sciences. Hawaii, 2003:220-229

[3] Li Zhen-hua, Chen Gui-hai, Qiu Tong-qing. Partition Nodes: Topologically-critical Nodes of Unstructured Peer-to-Peer Networks[J]. Journal of Software, 2008, 19(9):2376-2388

[4] Liu X, Xiao L, Kreling A, et al. Optimizing Overlay Topology by Reducing Cut Vertices[C]// Proceedings of ACM NOSSDAV. 2006

[5] Hochman H M, Rodgers J D. Pareto Optimal Redistribution[J]. The American Economic Review, 1969, 59(4):542-557

[6] Yu Yi-jiao, Jin Hai. A Survey on Overcoming Free Riding in Peer-to-Peer Networks[J]. Chinese Journal of Computers, 2008, 31(1)

[7] Liu Yuhua, Yang Chun, Xu Kaihua, et al. Model of Controlling the Hubs in P2P Networks[J]. Journal of Networks, 2009, 4(4)

[8] Iamnitchi A, Ripeanu M, Foster I. Small-world file-sharing communities[C]// IEEE infocom. 2004

[9] Hui Y K, Lui C S, Yau K Y. Small-world overlay p2p networks [R]. CS-TR-2004-04. The Chinese University of Hong Kong, April 2004

[10] Kempe D, Dobra A, Gehrke J. Gossip-Based Computation of Aggregation Information[C]// Proceedings of IEEE FOCS. 2003

[11] 陈林书，柳媛慧. P2P 网络中基于节点能力自适应的搜索算法[J]. 湖南科技大学学报：自然科学版，24(2)

[12] 任超，李战怀，张英. 异构 P2P 网络的分布式查询协议[J]. 电子科技大学学报，38(1)

[13] Chawathe Y, Ratnasamy S, Breslau L, et al. Making gnutella-like P2P systems scalable[C]// Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. Karlsruhe, Germany, August 2003: 25-29