ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.3, March 2009, pp.660−670
DOI: 10.3724/SP.J.1001.2009.03226

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种结构化 P2P 协议中的自适应负载均衡方法*

熊 伟[1+], 谢冬青[2], 焦炳旺[1], 刘 洁[3]

[1](湖南大学 计算机与通信学院,湖南 长沙 410082)

[2](广州大学 计算机科学与教育软件学院,广东 广州 510006)

[3](广州大学 实验中心,广东 广州 510006)

Self-Adaptive Load Balancing Method in Structured P2P Protocol

XIONG Wei[1+], XIE Dong-Qing[2], JIAO Bing-Wang[1], LIU Jie[3]

[1](College of Computer and Communications, Hu'nan University, Changsha 410082, China)

[2](School of Computer Science and Educational Software, Guangzhou University, Guangzhou 510006, China)

[3](Experimental Centre, Guangzhou University, Guangzhou 510006, China)

+ Corresponding author: E-mail: Freeprogman@hotmail.com

Xiong W, Xie DQ, Jiao BW, Liu J. Self-Adaptive load balancing method in structured P2P protocol. *Journal of Software*, 2009,20(3):660−670. http://www.jos.org.cn/1000-9825/3226.htm

Abstract: This paper presents a self-adaptive load balancing algorithm, in which each node creates a local load distribution view with a passive load statistic method and a local file requested view with a file requested statistic method. When the load imbalance exists in the system, the heavily loaded node will make the logical links pointing to itself point to a lighty loaded node in its local load distribution view, with the indegree of the heavy loaded node decreasing and that of the lighty loaded node increasing, the load imbalance magnitude will decrease. When the request load of the heavy loaded node is high, the node will use its local file request view to get the popular file and cache the file to corresponding target node. Results from simulation experiments indicate that the system has a good load balance under Zipf-like requests distribution if it runs the self adaptive load balancing algorithm. To some extent, caching requires some extra messages, but fewer than the cache hit messages under some condition, so caching can reduce the overall load of the system.

Key words: peer to peer; load balancing; structured overlay; cache; Chord

摘 要: 提出一种自适应负载均衡方法,方法采用一种被动式结点负载统计方法生成局部负载视图;一种文件访问统计方法生成局部文件访问视图;当系统内结点负载存在差异时,重载结点把指向自身的逻辑链路迁移至指向局部负载视图中的轻载结点,通过减小重载结点入度和增加轻载结点入度来减小结点间负载差异;当结点的请求负载较高时,通过局部文件访问视图计算需要缓存的热点文件及目标结点,降低承载热点文件的结点请求负载.实验结果表明,在用户查询服从Zipf分布的环境下,自适应负载均衡方法可使系统负载达到较好的均衡;缓存方法虽然在一定程度上增加了缓存和更新开销,但在一定条件下比查询消息命中缓存节省的网络开销要小,降低了系统的整体负载.

* Supported by the National Natural Science Foundation of China under Grant No.60673156 (国家自然科学基金); the Key Project of the Ministry of Education of China under Grant No.105129 (教育部科学技术研究重点项目)

Received 2007-07-20; Accepted 2007-10-26

关键词: P2P;负载均衡;结构化覆盖网;缓存;Chord
中图法分类号: TP393 文献标识码: A

近年来,P2P 软件用户增长迅速,负载均衡已成为一个突出、需要解决的问题.由于在 P2P 环境下,单个结点无法准确了解全局负载分布信息和用户查询具有不平衡性,使平衡各结点负载成为一个难以解决的问题.

在结构化 P2P 协议中,各结点通过承载一定的键值空间来分担网络负载.理想情况下,结点和文件分布均匀,各文件访问概率基本相当,各结点负载均匀分布,系统具有良好的负载均衡并能发挥最优性能,然而在实际网络中,事实并非如此,基于 DHT 算法的 P2P 系统存在严重的负载不平衡[1].引起负载不平衡主要有 3 个原因:(1) 各结点承载的键值空间相差很大.在大规模的结构化 P2P 系统中,基于 DHT 算法生成结点 ID 的方法使结点承载的最大键值空间是最小键值空间的 $O(\log N)$倍,例如 Chord[2],Pastry[3]等 P2P 协议;(2) 各结点承载的文件数目相差较大.即使各结点承载的键值空间相同,各结点负载也不一定均匀.由经典球盒问题[4]可知,在具有 n 个结点和 n 个键值的情况下,将有$\Theta(\log N/\log\log N)$个键值以很高的概率被分配到同一个结点上;(3) 用户查询具有不平衡性.目前提出的结构化 P2P 模型均假设用户查询随机而均匀.研究发现,P2P 系统中的查询请求分布和 HTTP 请求分布具有很大的相似性,均服从 Zipf 分布定律[5].各关键字查询的概率存在数量级差异,承载热点文件的结点负载将远大于其他结点.

目前有较多的方法采用虚拟服务器(virtual server)[6−9]或多 Hash 方法[10,11]来均衡系统内结点负载,这些方法主要针对前两个原因引起的负载不平衡,有以下不足:(1) 方法通过虚拟服务器迁移或多 Hash 方法调整结点承载的键值空间或文件数目以实现负载均衡,没有考虑用户查询的不平衡性.即使这些方法可以实现单个结点只承载一个热点文件,承载热点文件的结点负载仍然远大于一般结点,不能解决 Zipf 查询下的负载均衡问题;(2) 某些方法是在结点超载时再执行负载均衡算法,是一种较为被动的平衡方法.当结点超载时再执行平衡算法,平衡算法开销有可能加剧超载结点的过载程度,使算法响应速度变慢;(3) 算法仅考虑瞬间的负载信息,不考虑负载历史信息,不能做出较为合理的分配策略.当结点负载波动较大时,平衡算法需要反复执行,网络开销较大.

针对负载均衡方法存在的不足,应对用户查询的不平衡性,本文提出一种自适应负载均衡方法,通过逻辑链路迁移与缓存相结合来达到负载均衡,具体包括:(1) 提出一种局部负载统计方法,包括路由表指针和反向结点的请求负载和路由负载,这种负载统计方法采用消息夹带结点负载信息的方式传递,不会增加网络消息数量;(2) 逻辑链路迁移是把指向重载结点的逻辑链路迁移,使其指向轻载结点,即流向重载结点的数据流转为流向轻载结点;(3) 逻辑链路迁移并不能降低热点文件被查询的次数.当用户查询的不平衡性很大,例如服从 Zipf 分布时,承载热点文件的结点负载将明显高于其他结点.为减轻存有热点文件的结点负载,本文在查询热点文件的路径上缓存热点文件,使查询消息不需要到承载热点文件的结点就可以找到该文件,减小承载热点文件结点的查询负载.本文通过文件访问历史信息统计来计算需要缓存的文件和目的结点,这种缓存方式提高了热点文件被缓存的概率和查询命中缓存的概率;(4) 提出一种分布式“拉”模型缓存的更新方式,这种“拉”模型更新方式使缓存更新不必在文件的 *successor* 中完成,避免热点文件的 *successor* 需要处理较多的更新消息.

实验结果表明,在用户查询极不平衡的环境下,逻辑链路迁移和缓存相结合的负载均衡方法不但可以使系统有较好的负载均衡,同时还具有以下特点:(1) 网络结点间负载稍有不平衡,逻辑链路迁移或缓存便可能发生,自适应网络结点负载变化.相对于结点超载时再执行平衡算法的方法而言,自适应负载均衡方法的响应速度快,增强了网络的稳定性;(2) 由于用户查询具有随机性,结点负载波动较大,单纯用瞬间负载衡量结点负载是不合理的.逻辑链路迁移和缓存触发条件均考虑结点的历史负载信息;(3) 缓存机制在一定程度上增加了缓存消息和缓存更新的网络开销,但这种开销比查询消息命中缓存而降低的网络开销要小,说明缓存机制在一定条件下可以降低系统整体负载.

1 基本概念

为方便讨论,引出几个本文将使用的基本概念.

定义 1(请求负载(request load)). 当文件查询消息到达目的结点,目的结点发送响应消息(简称为应答消息)给查询用户时,目的结点的请求负载就增加 1.请求负载即为单位时间 T 内结点发送的应答消息数量.

定义 2(路由负载). 除发送应答消息之外,发送其他消息均属于路由负载.这部分负载包括查询消息路由、网络维护、与缓存机制相关的消息路由等.在实际网络中,缓存与缓存更新应答消息比一般消息要大,本文把一般消息权值设为 1,缓存消息和缓存更新消息权值分别设为 w_c,w_u(w_c,w_u 均大于等于 1).路由负载即为单位时间 T 内结点处理的消息(应答消息除外)加权和.

定义 3(负载). 单位时间 T 内结点请求负载和路由负载的加权和.一般情况下,应答消息比一般消息要大,应答消息权值设为 w_r.所以负载计算公式为:请求负载×w_r+路由负载.把结点负载分为请求负载和路由负载的原因在于:当结点的请求负载远大于其他结点的请求负载时,该结点很可能承载了热点文件;当结点路由负载高时,该结点的入度较大或者该结点的邻居结点承载了热点文件.

定义 4(反向结点). 假设结点 a 的路由表中含有指向结点 b 的路由表指针,则称 a 为 b 的反向结点.

2 自适应负载均衡方法

路由负载与入度有密切的关系,结点入度越大,结点的路由负载越大.目前,典型结构化协议在选取结点作为路由表指针时没有考虑结点负载与入度大小,结点的入度相差较大,使结点间的路由负载有较大差异.为减小结点间的负载差异,本文通过一种逻辑链路迁移的方法,把链路从重载结点迁移给轻载结点,一方面减少重载结点的路由负载,另一方面增加了轻载结点的路由负载,从而降低结点间的负载差异.

在典型结构化 P2P 协议[2,3]中,对于给定的关键字查询消息,无论发起查询关键字消息的是哪个结点,该关键字查询的目的结点总是唯一的.图 1 是一个 Chord 网络,假设关键字 10011 为查询热点,$N19$ 结点请求负载将远高于其他结点,为降低其请求负载,必须使查询消息在到达结点 $N19$ 之前找到相应的热点文件.因此有必要在查询消息的路经结点缓存热点文件,使查询消息不必到达目的结点就可以找到热点文件,从而减轻承载热点文件的结点的请求负载,减小结点负载差异.

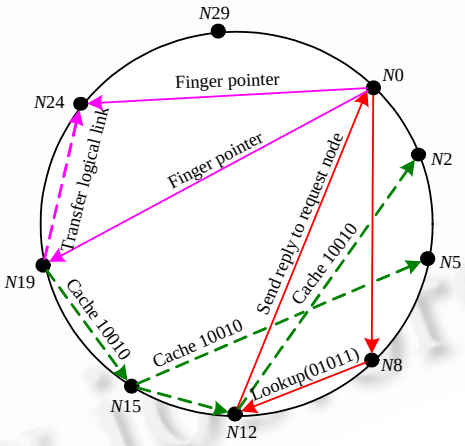

Fig.1 Demonstration of the load balancing method

图 1 负载平衡方法示意图

自适应负载均衡方法即由逻辑链路迁移和缓存相结合来实现结点负载均衡.本文将在第 2.1 节详述结点如何进行负载统计,根据统计方法产生的局部负载视图,结点可以判断自身负载轻重;在第 2.2 节详述逻辑链路迁移方法,回答什么时候发生逻辑链路迁移及链路迁移到哪个结点;在第 2.3 节详述文件访问统计方法,根据文件访问统计产生的局部文件访问视图,缓存算法回答什么时候应该缓存、缓存哪个文件到哪个目标结点,此外,在第 2.3 节还将详述一种新的缓存更新机制.

2.1 局部负载统计

对于任意结点,若其知道系统负载分布的情况越多,其做出的抉择相对也越合理.考虑到不增加网络开销,每个结点在发送消息时,均把自身负载信息加入到消息中.负载信息 *Load* 的数据结构包含如下成员:结点 *node*,实际系统中是结点地址;结点 *node* 的请求负载 *requestLoad*;结点 *node* 的路由负载 *routingLoad*;结点 *node* 的总负载 *load*,*load*=*requestLoad*×w_r+*routingLoad*;结点 *node* 的入度 *indegree*;记录收到反向结点 *node* 的消息的最近时间,用 *lastArriveTime* 表示.

对于任意结点 *n*,其统计的负载信息包括以下 3 个方面:(1) 结点本身负载信息列表 *loadList*,元素为 *Load* 结构类型.结点每隔周期 *T* 计算一次自身请求负载、路由负载和结点入度,并加入列表.(2) 反向结点的负载信息 *counter_node_load_list*,元素为 *Load* 结构类型.结点 *b* 收到结点 *a* 的消息后,如果反向结点列表中含有 *a* 的负载信息则更新,否则添加一条记录,并记录当前时间,删除列表中过期的负载信息.(3) 路由表结点的负载信息 *finger_node_load_list*,元素为 *Load* 结构类型.随着稳定算法(finger stabilization algorithm)更新路由表而相应更新.如图 1 所示,当结点 *N*0 更新第 5 个路由表入口时,假设由结点 *N*19 响应,结点 *N*19 在响应消息中夹带自身平均负载信息发送给结点 *N*0,*N*0 收到消息后更新 *finger_node_load_list*.

定义 4(结点平均负载). 结点本地负载信息表 *loadList* 中最近的 *x* 个周期的负载信息平均值用 L_s 表示,L_s.*load* 表示平均负载因子,L_s.*requestLoad* 表示平均请求负载,L_s.*routingLoad* 表示平均路由负载,L_s.*indegree* 表示平均入度.结点在发送消息给其他结点时,把平均负载 L_s 加入消息发送给目的结点.

定义 5(局部平均负载). 路由表指针的负载信息列表(*finger_node_load_list*)和反向结点负载信息列表(*counter_node_load_list*)中各结点负载的加权平均值.局部平均负载用 L_{avg} 表示(L_{avg}.*load* 表示局部平均负载因子,L_{avg}.*requestLoad* 表示局部平均请求负载,L_{avg}.*indegree* 表示局部平均入度).

由上述可知,局部性负载信息统计是一种被动式的结点负载信息统计方式,负载信息采用稍带的方式进行传递,不增加网络消息数量.由于用户查询具有随机性,结点负载波动性较大,如果用瞬间负载来衡量结点负载可能导致负载统计偏差较大,因此,结点在传递自身负载信息时均采用本地结点平均负载 L_s.局部负载统计将产生一个局部负载视图,结点根据局部负载视图,可以方便计算出结点平均负载与局部平均负载,由此可判断自身负载轻重及明确局部结点负载分布.

2.2 逻辑链路迁移

结构化 P2P 协议[2,3]在查找关键字时一般均采用贪婪算法,因此,结点入度大小与结点的路由负载密切相关,结点入度越大,结点的路由负载越大,因此,本文逻辑链路迁移实际上是将指向重载结点的链路迁移到轻载结点,以减小结点间的负载不平衡.本文算法以 Chord 为基础,原有的 Chord 稳定算法在选择结点作为路由表指针时没有考虑结点负载信息,本文提出一种新的 Chord 稳定算法,描述如下:

逻辑链路迁移算法.

n.fix_fingers()

1: *i*=Zipf random *index*>1 into *finger*[ ];
2: if (*i*<=*m*) *right*=*finger*[*i*+1].start;
3: else *right*=*n*.*id*;
4: if (*finger*[*i*].*node*∉[*finger*[*i*].*start*,*right*))
5: *finger*[*i*].*node*=*find_successor*(*finger*[*i*].*start*);
6: else
7: *tmp*=*finger*[*i*].*node*;
8: *finger*[*i*].*node*=*tmp*.*select_node*(*finger*[*i*].*start*,*right*);

n.select_node(*left*,*right*)

9: if (L_s.*load*∗*p*<L_{avg}.*load*) or (L_s.*indegree*<L_{avg}.*indegree*/2)

```
10:          return n;
11:    min=L_s;
12:    for each tmp∈(finger_node_load_list∨counter_node_load_list)
13:        if(tmp.node∈[left,right) and tmp.load<min.load)
14:            min=tmp;
15:    return min.node
```

(1) 首先用 *Zipf* 分布(*a*=1)选取一个路由表入口 *i* 进行更新,入口编号越大,选取概率越大(语句 1).

(2) 计算区间 *finger*[*i*].*interval* 的右值(语句 2,3).如果当前路由表指针指向的结点 *finger*[*i*].*node* 不在区间 *finger*[*i*].*interval* 中(语句 4),则按 Chord 方式进行更新(语句 5).否则,执行第 3 步;

(3) 直接发送消息给当前 *finger*[*i*].*node*,*finger*[*i*].*node* 收到消息后,按规则在本地选择一个更适合作为路由表指针的结点(语句 7,8),具体方法是:首先判断结点平均负载 L_s 与局部平均负载 L_{avg} 之间的关系,当满足关系 $L_s.load*p(p\geq 1)$小于 $L_{avg}.load$,或者 $L_s.indegree$ 小于 $L_{avg}.indegree/2$ 时,则返回本地结点(语句 9,10),否则执行第 4 步.$L_s.load*p$ 小于 $L_{avg}.load$ 表明本地结点属于轻载结点,如果把链路迁移,则更不利于负载均衡.$L_s.indegree$ 小于 $L_{avg}.indegree/2$ 表明结点的入度过低,即使结点近期负载较高,也很可能是查询不平衡所致,此时迁移链路,则该结点负载起伏较大,因此也不适合链路迁移.

(4) 在反向结点和路由表指针中查找属于区间 *finger*[*i*].*interval* 并且负载最小的结点(语句 11~15),发送消息给负载最小结点.如果属于区间 *finger*[*i*].*interval* 的负载最小结点为本地结点,则不用发送消息.

(5) 负载最小的结点计算本地平均负载加入消息,发送给消息原结点,原结点更新第 *i* 个入口指针(语句 8),并更新 *finger_node_load_list* 负载信息.

如图 1 所示,结点 *N*0 在更新第 5 个路由表入口时,*N*19 在区间[10000,00000]中,所以直接发送消息给结点 *N*19,*N*19 收到消息后,发现满足迁移条件,并且 *N*24 是区间[10000,00000]中负载最小的结点,因此更新消息发送给结点 *N*24,*N*24 把自身平均负载信息加入消息后发送给原结点 *N*0,则 *N*0 的第 5 个路由表指针 *finger*[5].*node* 为 *N*24;由算法可知,当 *finger*[*i*].*node*$\notin[n+2^{i-1},n+2^i]$时,计算复杂度与 Chord 一致,为 $O(\log N)$,否则,跳计数至多为 3,所以新维护算法开销比原维护算法开销要小.可以看出,逻辑链路迁移实质上是把结点路由表中指向重载结点的逻辑链路转移,使其指向局部负载视图中的轻载结点,通过降低重载结点的入度来减小重载结点的路由负载,通过提高轻载结点的入度来增大轻载结点的路由负载,从而降低重载结点与轻载结点间的负载差异,其本质上是通过局部负载均衡来实现整体负载均衡,但是逻辑链路迁移只能改变结点的路由负载,并不能改变结点的请求负载,当用户查询的不平衡程度很大时,请求负载就成为结点负载不平衡的重要因素.

2.3 缓 存

2.3.1 文件访问统计

由于用户查询具有随机性,因此仅凭短期内的查询次数并不能说明文件的查询热度,因此必须跟踪每个文件的访问历史信息.结点包含的单个文件信息用 *fileInfo* 表示,其与缓存相关的数据结构成员包括:文件 ID,用 *fid* 表示;周期文件访问计数器 *requestTimes*;文件访问历史数据列表 *history*,每隔 *T* 时间段把文件访问计数器 *requestTimes* 插入列表前端,然后 *requestTimes* 设为 0;记录单个周期 *T* 内访问该文件的反向结点及该反向结点访问次数的列表 *requestNbrs*,其(*key*,*value*)对即为(反向结点,访问次数),如图 1 所示,*N*0 查询关键字 01011,则结点 *N*12 符合查询要求,反向结点为 *N*8,更新结点 *N*8 在本周期的访问次数;更新源 *cacheSource*,如图 1 所示,当结点 *N*12 发送缓存消息给 *N*2,*N*2 保存该文件,并把该文件更新源设为 *N*12;文件 *fid* 的最近一次更新时间为 *lastUpdateTime*.当结点需要缓存当前文件 *fid* 到其他结点时,选择 *requestNbrs* 取出最近一个周期访问当前文件最多的反向结点.缓存文件到访问该文件次数最多的反向结点,可以增加查询命中缓存的概率,减少网络开销.

结点文件表(FileTable)包含有两种文件列表,一是映射到结点的文件列表(*file_list*),二是缓存文件列表(*cache_list*),列表元素均为结构 *FileInfo* 对象.当结点执行缓存算法时,算法在 *file_list* 和 *cache_list* 中选择文件访问历史数据列表 *history* 中最近 *z* 个周期平均访问次数最高并且当前周期内的文件访问计数 *RequestTimes* 不为

0 的文件,前一个条件要求该文件相对其他文件来说,最近查询的次数最高,是一个热点文件;由于存在缓存,当文件缓存后,该文件的访问次数将迅速下降,甚至没有查询该文件的消息会查询到该结点,*RequestTimes* 大于 0 表示该文件还可以进行缓存,本文在缓存算法中采用 *fileTable.GetCacheFileInfo*()函数表示该功能.

文件访问统计将产生一个局部文件访问视图,根据局部文件访问视图,结点可以明确文件的查询热度.当结点请求负载在局部负载统计产生的局部负载视图中较高时,结点根据局部文件访问视图计算需要缓存的文件和目的结点.

2.3.2 缓存算法与缓存更新

缓存算法是周期性执行的一种算法,描述如下:

缓存算法.

n.cache_periodic()

1: if (L_s.*requestLoad*>L_{avg}.*requestLoad*×*k*) and (L_s.*load*>*q*×L_{avg}.*load*)

2: *fileInfo=fileTable.GetCacheFileInfo*();

3: if (*fileInfo==null*) return;

4: *dest=fileInfo.GetCacheDestination*();

5: *dest.AddCacheFile*(*fileInfo.fid*);

(1) 文件缓存的触发条件是:结点请求负载平均值 L_s.*requestLoad* 是局部平均请求负载 L_{avg}.*requestLoad* 的 *k*(*k*>1)倍以上(L_s.*requestLoad*>L_{avg}.*requestLoad*×*k*),并且结点平均负载因子是局部平均负载因子的 *q*(*q*≥1)倍以上(L_s.*load*>*q*×L_{avg}.*load*)才执行缓存算法.前者要求当前结点的近期平均请求负载远大于局部平均请求负载,相当于要求当前结点必须承载了热点文件;后者要求当前结点在局部负载视图中表现为重载结点.

(2) 从 *fileTable* 中选取最佳需要缓存的热点文件,如果不存在这样的文件,则返回(语句 2,3).

(3) 计算热点文件应该缓存的反向结点,并发送缓存消息给目标结点(语句 4).*fileInfo.GetCacheDestination*()返回对象 *fileInfo* 的 *requestNbrs* 列表中最近一个周期访问当前文件最多的反向结点.

(4) 结点收到缓存消息,把文件加入缓存列表 *cache_list*,并设置 *lastUpdateTime* 为当前时间,如果缓存超过一定数量 *C*(*C* 为缓存大小),则删除近期访问量最小的缓存文件(语句 5).

如果文件缓存副本更新均在相应文件的 *successor* 上完成,则承载热点文件的 *successor* 需要处理较多的更新消息.为避免这种情况的发生,本文采用一种“拉”模型分布式更新算法,其中 *n.GetTime*()返回当前时间;*n.Contains*(*fileInfo.fid*)判断当前结点的 *fileTable* 中是否存有 *fileInfo.fid* 文件的相关信息.算法描述如下:

缓存更新算法.

n.cache_update_periodic()

1: for each *fileInfo*∈*fileTable.cacheList*

2: if (*fileInfo.lastUpdateTime*+T_u<*n.GetTime*())

3: *tmpNode=fileInfo.cacheSource*;

4: while (not *tmpNode.Contains*(*fileInfo.fid*))

5: *tmpNode=tmpNode.find_successor*(*fileInfo.fid*);

6: // *tmpNode* sends the *fileInfo* to *n*!

7: // *n* receives message and updates its *fileTable*

(1) 结点 *n* 定时(隔周期 *T*)取得需要更新的每一个文件(语句 1,2,假设缓存文件过期时间均为 T_u,*T*<<T_u).

(2) 结点 *n* 发送更新请求消息给缓存文件的更新源 *cacheSource*(语句 3).如果更新源 *cachesource* 中没有该文件数据,则执行 *find_successor*(*fid*),直到在查找的过程中发现某结点存有相关文件数据(语句 4,5).

(3) 目标结点 *tmpNode* 发送更新数据给结点 *n*.结点 *n* 更新相关缓存文件,并把该缓存文件属性 *lastUpdateTime* 设为当前时间,文件属性 *cacheSource* 设为结点 *tmpNode*(注释语句 6,7).

这种分布式数据更新方法使热点文件的更新不必在该文件的 *successor* 中完成.如图 1 所示,假设结点 *N*19

存有热点文件 10010,运行中热点文件 10010 首先缓存到结点 *N*15,*N*15 把文件缓存到结点 *N*12 和 *N*5,*N*12 把文件缓存到结点 *N*2.结点更新文件 10010 时,*N*2 在 *N*12 中取数据,*N*12 和 *N*5 在 *N*15 中取数据,*N*15 在 *N*19 中取数据,形成一个树型结构.当结点 *N*12 把 10010 删除后,结点 *N*2 更新时,仍然发送消息给 *N*12 结点,结点 *N*12 在其文件列表中没有发现 10010,则执行 *find_successor*(10010)发送消息给下一跳 *N*15,*N*15 保存有 10010 的数据,*N*15 发送最新数据给结点 *N*2,*N*2 记录文件 10010 的更新源 *cacheSource* 为 *N*15 并设置文件 10010 的 *lastUpdateTime* 为当前时间,更新结束.

3 模拟实验与分析

实验主要包括以下几个步骤:(1) 目前有较多平衡方法的主要目的是平衡各结点承载的关键字,因此,首先考察关键字分布均匀、用户查询随机均匀的负载分布,参数设置见表 1,实验时间为 2 小时;(2) 考察用户查询的不平衡性对结点负载分布的影响,实验时,Zipf 分布参数 a=1,其他参数与第 1 步相同,实验时间为 2 小时;(3) 重复第 2 步 Zipf 查询实验,然后启动缓存和逻辑链路迁移算法运行 2 小时.与自适应负载均衡方法相关的参数设置见表 2.

Table 1 Basic experiment parameters
表 1 基本实验参数

Parameter	Description	Value
N	Number of nodes	1 000
n_f	Number of files on each node	5
T_r	Searching file interval	30s
T	Computing load interval	1m
w_r	Reply message weight	2
T_f	Finger stabilization interval	30s
T_s	Successor stabilization interval	30s
T_p	Finger ping interval	10m

Table 2 Balancing algorithm parameters
表 2 均衡算法参数

Parameter	Description	Value
x	Balancing coefficient	3
p,q	Correction coefficient	1.1
z	Balancing coefficient	5
k	Cache coefficient	5
T_u	File updating interval	30m
w_c	Weight of caching messages	2
w_u	Weight of updating messages	2
c	The cache size	10

图 2 为查询随机均匀下负载最高结点 *N*593 与负载最低结点 *N*861 的负载变化图.由于关键字分布均匀并且查询均匀,各结点的请求负载基本相当,但结点 *N*593 的入度最大,为 29,负载平均值为每分钟 79.7 条消息,而结点 *N*861 的入度最小,为 2,负载平均值为每分钟 23.2 条消息.实验结果表明,在一个查询随机均匀的环境下,单纯分布相同的关键字平衡方法并不能使结点负载达到平衡.从理论上来说,结点 *N*861 在结点 *N*593 承载的空间中插入虚拟服务器可以使两结点负载达到基本平衡,即采用虚拟服务器技术可以解决查询均匀环境下的负载平衡.图 3 为查询服从 Zipf 分布(a=1)时的负载最高结点 *N*443 与负载最低结点 *N*861 的负载变化图.由于结点 *N*443 承载了热点文件,其请求负载远高于其他结点,使其总体上成为新的负载最高结点,平均值为每分钟 276.7 条消息.结点 *N*443 的负载波动很大,每分钟处理消息可相差 98 条以上,Zipf 查询使结点负载波动更明显,如果用瞬时负载信息作为结点负载,则结点有可能在某一时刻表现为重载结点,而在另一时刻却表现为轻载结点,因此,采用瞬间负载来作为均衡算法的触发条件,则均衡算法可能需要反复无谓地执行,显然用瞬时负载信息来衡量结点负载是不合理的.图 4 为平衡后 1 小时内(180 分钟~240 分钟)平均负载最大结点和平均负载最小结点的对比图,与图 3 相比,负载最高结点为 *N*110,负载平均值为每分钟 34.5 条消息,负载最低结点为 *N*489,负载平均值为每分钟 23.4 条消息.负载最高结点在单个周期内并不总是大于负载最低结点,因此,在时间段内(180 分钟~240 分钟)表现为负载最高的结点不一定在下一时间段内仍然为负载最高结点.负载均衡后,由于缓存触发条件较为苛刻,此时基本没有缓存消息,一段时间内,各结点的请求负载表现变化不大,但链路迁移时时刻刻都有可能发生,以适应结点负载变化.

图 5 为结点平均负载分布图.在查询均匀(a=0)的环境下,80%以上的结点平均负载在每分钟 30 条~50 条消息之间,1%的结点负载平均值大于 60.Zipf 查询(a=1)下,与查询随机均匀的曲线相比,约 83%的结点负载有所降低,1%的结点负载平均值大于 100,个别结点平均负载在 200 以上,用户查询不平衡使结点负载不平衡的程度加剧.平衡算法执行后,99%的结点负载在 25~33 之间,最大平均负载为 35.4,是最小值的 1.45 倍.

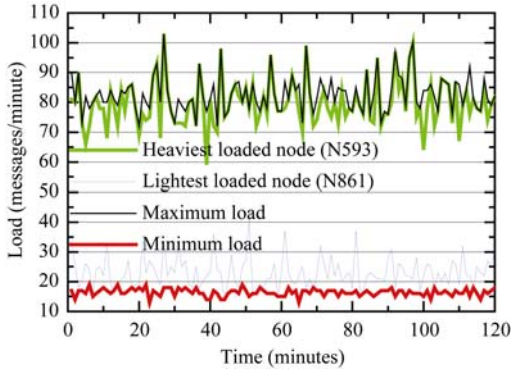

Fig.2 Load of the heaviest loaded node and the lightest loaded node (Zipf'a=0)

图 2 最大负载结点与最小负载结点的负载变化图(Zipf 参数 a=0)

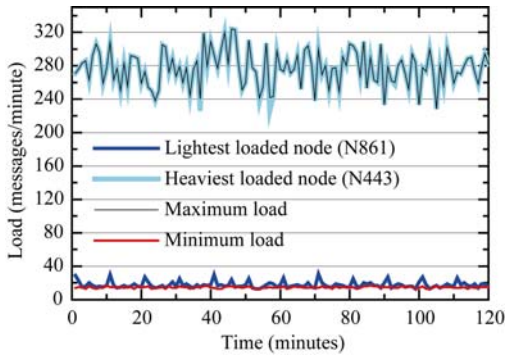

Fig.3 Load of the heaviest loaded node and the lightest loaded node (Zipf'a=1)

图 3 最大负载结点与最小负载结点的负载变化图(Zipf 参数 a=1)

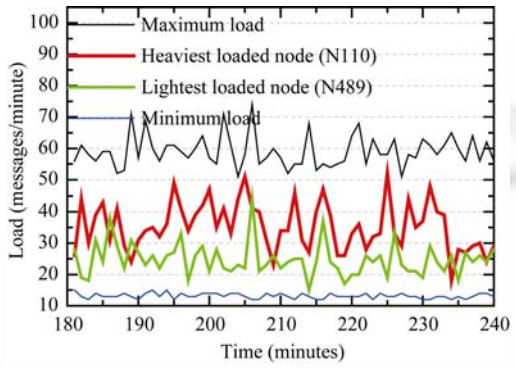

Fig.4 Load of the heaviest loaded node and the lightest loaded node (Zipf'a=1)

图 4 最大负载结点与最小负载结点的负载变化图(Zipf 参数 a=1)

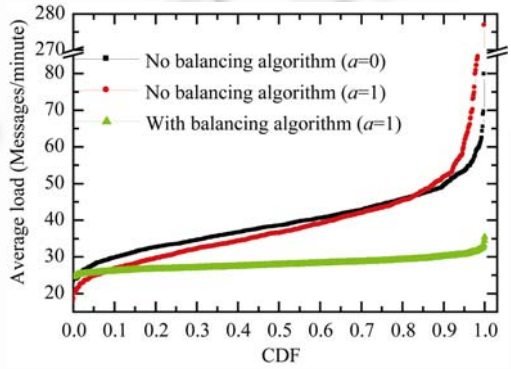

Fig.5 The average load of nodes

图 5 平均总负载对比

图 6 为周期最大负载和周期最小负载的变化图.结点 $N443$ 在平衡算法启动前为负载最高结点,平衡算法启动后,一方面,把热点文件缓存到前驱 $N442$ 等结点,另一方面,通过逻辑链路迁移,把指向 $N443$ 的指针转移到局部负载视图中的轻载结点,使 $N443$ 的负载迅速降低.实验中,结点 $N442$ 也是一个重载结点,由于其缓存了结点 $N443$ 的热点文件,使得该结点在单个周期内负载有所上升,这是负载在平衡过渡期中的波动现象.从图中可以看出,当平衡算法启动后,周期最大负载迅速降低并很快达到稳定,算法响应速度较快.

图 7 为平衡算法启动后与缓存相关的各类消息速率图.平衡算法启动后,请求负载高的结点把热点文件缓存到其他结点,但随着时间的推移,缓存消息(cache messages)逐渐下降,1 小时后基本上没有.相对于缓存开销(包括缓存、缓存更新请求及更新应答消息)而言,命中缓存的消息数量远大于三者之和,这说明,在一定条件下,缓存可以降低系统整体负载.

表 3 为结点总数不同的情况下的实验结果,从表中可以看出,平衡算法启动之前,参与结点越多,结点负载差异越大,主要是因为在 Zipf 查询下,文件数量越多,各文件的查询概率相差越大,使结点请求负载和路由负载差异变大.平衡算法启动后,结点最大负载平均值不超过最小负载平均值的 1.7 倍,均能达到较好的平衡效果.当参与结点数量增大时,结点平均负载均有升高,这是因为 DHT 查找跳计数和系统维护开销有所增大;从平衡前后的结点平均负载可以看出:平衡后的结点平均负载均小于平衡前的结点平均负载,主要原因在于:命中缓存减少的网络开销相对于缓存引起的开销要大得多;平衡前后结点平均负载之差随结点数的增大而有所增大,主要原

因在于:当文件数增大时,Zipf 查询下各文件热度相差增大,热点文件查询次数增多,查询消息命中缓存的概率增大,使系统整体负载降低增大.表 4 为缓存系数 k 取不同值的实验结果.当 k>5 时,k 值越大,缓存触发条件越高,使结点请求负载差异较大,结点间负载差异越明显.表 5 为分布参数 a 取不同值时的实验结果.当 a 增大时,系统整体负载有下降趋势,主要原因在于:分布参数 a 越大,热点文件查询概率越高,查询命中缓存的概率越高,使系统整体负载降低越大.表 6 为 w_c,w_u,w_r取不同权值时的实验结果.由于结点进行缓存的触发条件是该结点的请求负载是其他结点的 k(k=5)倍,即当这些消息权值增大时,结点间的负载差异将增大.此时,通过减小 k 降低缓存触发条件可以降低结点间负载差异.

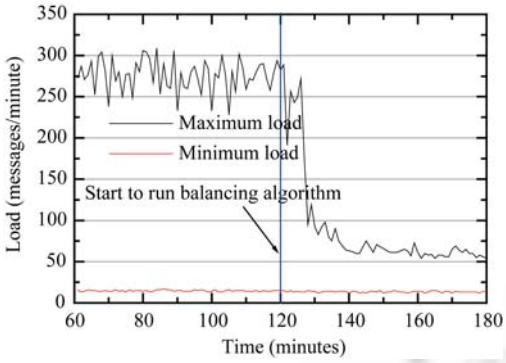

Fig.6 Maximum load and the minimum load in the experiment(*Zipf*'*a*=1)

图 6 结点负载最大值和最小值变化图(Zipf 参数 a=1)

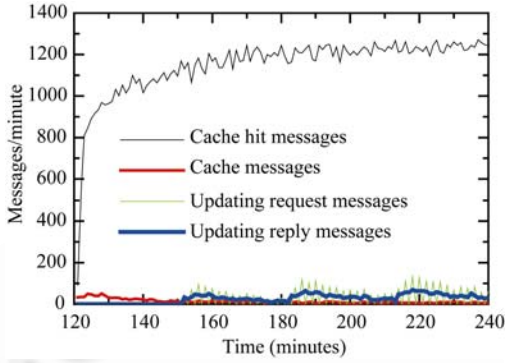

Fig.7 Velocity of cache hit messages, cache messages, updating request messages and updating reply messages

图 7 缓存命中消息、缓存消息、更新请求消息与更新应答消息的速率图

Table 3 Simulation result under different number of nodes

表 3 结点数不同的实验结果

Number of nodes	No balancing algorithm		
	Min.	Avg.	Max.
2000	16.8	40.7	489.8
3000	16.5	41.7	701.3
4000	15.8	43	875.4
Number of nodes	With balancing algorithm		
2000	24.6	28.6	37.7
3000	24.3	29.3	40.4
4000	24.9	29.7	40

Table 4 Simulation result under different cache coefficient

表 4 缓存系数取不同值的实验结果

k	Min.	Avg.	Max.
2	23	26.3	32.9
3	23.5	26.8	32.3
4	22.9	27.3	33
5	23.4	27.7	34.5
6	23.8	28.1	38.9
7	24.5	28.3	41.9
8	24.2	28.5	50.5

Table 5 Simulation result under different Zipf parameter

表 5 Zipf 分布参数不同的实验结果

a	Min.	Avg.	Max.
0.8	25.5	29.2	36.9
1	23.4	27.7	34.5
1.5	20.0	23.6	29.3
2	17.6	20.7	27.4

Table 6 Simulation result under different weight of messages

表 6 消息权值不同的实验结果

w_c,w_u,w_r	Min.	Avg.	Max.
2	23.4	27.7	34.5
5	26.5	33.4	56.3
8	29.2	39.5	65.2
10	30.5	43.5	81.8

4 其他问题

假设有一种极端的环境:每个结点缓存了所有结点的文件,查询消息都可以在本地结点中完成,各结点负载可以实现基本相同.然而在实际网络中,结点存储和计算能力均是有限的,因此这是不现实的,即使缓存可以使各结点请求负载基本相当,但路由负载却可能相差很大.目前研究表明,在用户查询极不平衡的情况下,仅采用缓存不能解决负载均衡问题[13,14].

结点很容易统计单个缓存文件的查询命中次数,当用户查询速率增大时,单位时间命中缓存的次数也将增大,因此,当系统查询负载增大时,缓存往往能够降低系统负载.由实验结果图 7 可知,即使更新周期为 10 分钟,命中缓存只降低 1 跳,查询命中缓存次数仍然远大于缓存更新开销,但如果所有的结点不再发送查询请求,则缓存命中次数为 0,此时缓存却增加了系统负载.因此,在实际中,当一段时间内结点发现某个缓存文件命中次数给网络减小的开销小于缓存引起的开销,结点可以根据策略删除该缓存文件.实验中各结点承载的文件数均相等(n_f=5),可以采用文献[7,10]中的方法实现各结点承载的文件数基本相当.结点 churn 对于系统整体负载分布影响不大,因此,实验没有考虑结点 churn 对负载分布影响.

目前,典型的结构化协议均可以实现链路迁移和缓存,例如 Pastry,Kademlia,TaPestry 等协议,这些协议均可以采用自适应负载均衡方法来平衡系统内的结点负载.

5 相关工作

文献[6]通过虚拟服务器来重新划分结点承载的键值空间及关键字,其实质上是通过频繁的结点加入和退出网络来达到各结点负载均衡,增加了网络的波动性,开销较大.文献[7]提出 3 种通过虚拟服务器迁移实现结点负载均衡的不同方法,即一对一、一对多和多对多模式.其基本思想是把重载结点的虚拟服务器转移给轻载结点以减少结点间的负载不平衡.文献[8]扩展了文献[7]一对多和多对多模式,使其适应网络动态性和异构环境.文献[9]也是一种采用虚拟服务器的方法.文献[10]采用多 Hash(d≥2)方法,使对象索引存储到负载最低的映射结点,其他 d–1 个映射结点只存储指向存储对象索引的结点的重定向指针,这种方法可以实现各结点承载的关键字数量有较好的平衡.文献[11]采用与文献[10]相似的方法.文献[12]在原始 P2P 结构之上建立 k-arg 树,用 k-arg 树来收集、发布结点负载信息以及生成虚拟服务器迁移策略.以上这些方法均主要考虑资源和键值空间的分布不平衡,没有考虑用户查询的不平衡性.当查询服从 Zipf 分布时,对于承载了热点文件的结点,决定其负载主要在于请求负载,使用虚拟服务器或者多 Hash 技术并不能降低热点文件的查询次数,热点文件被分配到哪个结点,则该结点将成为重载结点,因此不能应对 Zipf 查询对负载分布的影响.

文献[13]利用 Pastry 路由表指针可替换的特点,在消息路由的过程中,路经结点根据消息中携带的结点与负载信息判断是否更新路由表.由于这种方法以长时间段内结点处理的消息总数作为负载,这是不合理的,因为结点处理的消息总数相等并不表示结点在单位时间内的负载仍然相当,且没有考虑缓存文件更新.文献[14]采用路由表冗余和缓存相结合的方法.当消息路由时,结点并不采用贪婪式算法,而是把消息发送给一个负载较轻的结点,但文献并没有说明如何建立和维护路由表冗余指针及其负载信息,也没有考虑缓存更新.文献[15]根据文件访问历史信息,通过结点变换位置给不同能力的结点分配相应的文件数量,同样不能应对 Zipf 查询引起的负载不平衡.当结点负载过重时,结点与邻居结点协商承载的文件数量,如果协商不成功,则增大协商的邻居结点数量,是一种被动式递归平衡算法,这种方法虽然不使用虚拟服务器,但结点变换位置相当于重新加入网络,尤其是在协商的邻居结点很多时,开销很大.文献[16]是一种基于流言传播的负载均衡方法,其主要思想是:当结点负载过高时,通过流言传播方式把自身承载的文件复制到其他结点,通过降低重载结点请求负载来减小结点间的负载差异,但仅采用复制方法并不能解决 Zipf 查询下的负载平衡问题[13,14].

6 结 论

在关键字分布均匀、用户查询随机均匀的环境下,由于结点的入度相差较大,结点路由负载成为负载不平衡的主要因素.当用户查询不平衡程度很高时,由于文件的查询热度相差很大,请求负载成为负载不平衡的另一个重要因素.热点文件被分配到哪个结点,该结点将成为重载结点,因此虚拟服务器、多 Hash 方法不能解决 Zipf 查询下的负载均衡问题.自适应负载均衡方法通过局部负载统计、文件访问统计方法产生局部负载视图和局部文件访问视图,采用逻辑链路迁移和缓存相结合的方法来平衡各结点负载,这种方法可以使系统负载达到较好的平衡效果.缓存方法虽然在一定程度上增加了缓存和更新开销,但在一定条件下比查询消息命中缓存节省的网络开销更小,缓存可以降低系统的整体负载.

References:

[1] King V, Saia J. Choosing a random peer. In: Proc. of the 23rd Annual ACM Symp. on Principles of Distributed Computing (PODC 2004). New York: ACM Press, 2004. 125−130.

[2] Stoica I, Morris R, Karger D, Kaashoek F, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for Internet applications. In: Cruz R, Varghese G, eds. Proc. of the 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SigComm). New York: ACM Press, 2001. 149−160.

[3] Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Guerraoui R, ed. Proc. of the IFIP/ACM Int'l Middleware Conf. London: Springer-Verlag, 2001. 329−350.

[4] Raab M, Steger A. Balls into Bins-A simple and tight analysis. LNCS 1518. London: Springer-Verlag, 1998. 159−170.

[5] Handurukande SB, Kermarrec AM, Le Fessant F, Massoulie L, Patarin S. Peer sharing behavior in the eDonkey network, and implications for the design of server-less file sharing systems. ACM SIGOPS Operating Systems Review, 2006,40(4):359−371.

[6] Karger D, Ruhl M. Simple efficient load balancing algorithms for peer-to-peer systems. In: Voelkerm GM, Shenker S, eds. LNCS. Berlin: Springer-Verlag, 2005. 131−140.

[7] Karthik A R, Lakshminarayanan K, Surana S, Karp R, Stoica I. Load balancing in structured P2P systems. In: Kaashoek MF, Stoica I, eds. LNCS. Berlin: Springer-Verlag, 2003. 68−79.

[8] Godfrey B, Lakshminarayanan K, Surana S, Karp R, Stoica I Load balancing in dynamic structured P2P systems. In: Kaashoek MF, Stoica I, eds. Proc. of the IEEE Infocom. 2004. http://www.ieee-infocom.org/2004/Papers/46_4.pdf

[9] Li ZN, Xie GG. A load balancing algorithm for DHT-Based P2P systems. Journal of Computer Research and Development, 2006,43(9):1579−1585 (in Chinese with English abstract).

[10] Byers J, Considine J, Mitzenmacher M. Simple load balancing for distributed hash tables. In: Kaashoek MF, Stoica I, eds. LNCS. Berlin: Springer-Verlag, 2003. 80−87.

[11] Ledlie J, Seltzer M. Distributed, secure load balancing with skew, heterogeneity, and churn. In: Proc. of the IEEE Infocom. 2005. Washington: IEEE Computer Society, 2005. 1419−1430.

[12] Zhu Y, Hu Y. Efficient, proximity-aware load balancing for DHT based P2P systems. IEEE Trans. on Parallel and Distributed Systems, 2005,6(4):349−361.

[13] Bianchi S, Serbu S, Felber P, Kropf P. Adaptive load balancing for DHT lookups. In: Proc. of the ICCCN. 2006. http://members.unine.ch/pascal.felber/publications/ICCCN-06.pdf

[14] Anwitaman D, Roman S, Karl A. Query-Load balancing in structured overlays. In: Proc. of the 7th IEEE Int'l Symp. on Cluster Computing and the Grid. 2007. http://lsirpeople.epfl.ch/rschmidt/papers/Datta2007QueryLoad.pdf

[15] Xu Z, Bhuyan L. Effective load balancing in P2P systems. In: Proc. of the 6th IEEE Int'l Symp. on Cluster Computing and the Grid. Washington: IEEE Computer Society, 2006. 81−88.

[16] Zhu C, Liu Z, Zhang WM, Xiao WD, Yang DS, Xu ZN. Load balance based on rumor mongering in structured P2P networks. Journal of China Institute of Communications, 2004,25(4):31−40 (in Chinese with English abstract).

附中文参考文献:

[9] 李振宁,谢高岗.基于 DHT 的 P2P 系统的负载均衡算法.计算机研究与发展,2006,43(9):1579−1585.

[16] 朱承,刘忠,张维明,肖卫东,阳东升,徐振宁.结构化 P2P 网络中基于流言传播的负载均衡.通信学报,2004,25(4):31−40.

熊伟(1977－),男,江西丰城人,博士,主要研究领域为分布式计算.

焦炳旺(1974－),男,博士,主要研究领域为对等网络.

谢冬青(1965－),男,博士,教授,博士生导师,主要研究领域为算法分析与设计,信息安全.

刘洁(1980－), 硕士, 主要研究领域为信息安全.