异构结构化 P2P 网络负载均衡方案*)

于 婧 张建辉 汪斌强

(国家数字交换系统工程技术研究中心 郑州 450002)

摘 要 结构化 P2P 网络由于采用 DHT 算法导致节点存储资源的不均衡,当前解决方案都是假定节点容量及负载是均匀分布在系统中,而忽略了实际网络存在的节点异构性的影响。本文提出的考虑节点异构性的结构化 P2P 网络负载均衡方案提出了负载均衡的衡量标准——负载平滑度,采用基于相同资源描述符的资源整体转移方案,以节点的邻居节点为平衡范围,描述了系统在节点加入、离开,资源加入以及节点过载情况下的算法,使得整个系统逐步达到负载均衡。该方案充分考虑了实际网络中存在的异构问题。仿真实验表明,该方案有效地解决了异构 P2P 网络下的负载均衡问题。

关键词 对等网络,异构,负载均衡

Load Balancing Algorithm of Heterogeneous Structured P2P Network

YU Jing ZHANG Jian-hui WANG Bin-qiang

(National Digital Switching System Research Center, Zhengzhou 450002, China)

Abstract DHT algorithms used in structured P2P network lead to unbalance of storage in each node. The projects currently are on the conditions of uniformly distributed node capacity and load, while ignoring the effect of heterogeneous. The algorithms we proposed introduce an evaluation of load balancing, that is load smoothness. It describes the algorithms when nodes join, leave and overload with the main idea of transferring resources with the same key holistically within the scope of neighbors. It takes the heterogeneous problem into consideration and the results show the validity in load balancing of heterogeneous structured P2P network.

Keywords Peer-to-peer network, Heterogeneous, Load balancing

1 引言

基于 DHT 的结构化 P2P 网络(Structured P2P network)[1],如 Chord [2], Pastry [3], Tapestry[4]和 CAN [5],采用分布式哈希表算法对资源进行存储和定位,是一种快速高效的资源组织方式,因此成为 P2P 网络发展的方向。然而 DHT 的工作方式是通过统一的哈希函数将资源映射到资源标识符,并将其存储在节点标识符与资源标识符数值相近的节点上,而系统中节点的分布是随机的,从而每个节点负责的标识符空间的大小也是不同的,某些节点可能拥有大段的标识符空间而导致存储大量的数据。因此,这种资源存储方式必然会导致存储在节点上的资源的数量的不平衡。假定所有的资源大小是相同的,首先想到的解决方案就是采用地址空间均衡策略(Address-space balancing[6])来保证节点拥有的地址空间大小的平均来达到负载均衡的目的。然而,最近的测量表明[7]系统中各个节点从带宽、存储容量以及 CPU 处理能力都是不同的,简单的地址空间均衡并不能解决节点资源平衡的问题。

Dabek 在文献[8]中提出的 Virtual server 方案是一种兼顾节点存储容量的地址空间均衡方案。它的主要思想是每个物理节点依据自身的容量来模拟多个虚拟节点,每个虚拟节点在 DHT 中代表一个节点,虚拟服务器占据 DHT 中的地址空间,而每个物理节点对应多个虚拟服务器,所以一个物理节点对应多块地址空间,而这些地址空间可能是非连续的。物理节点占据的地址空间的大小与它所划分的虚拟节点个数相关,从而也就与该物理节点的容量相关联。在 Virtual server 方法中,数据的存储和路由是发生在虚拟节点级别而不是物理节点级别。负载均衡是通过从过载节点向和非过载节点移动虚拟节点来达到的[9]。

然而,Virtual server 需要根据自身的容量以及系统内总容量、系统内总节点数来确定所占用的地址空间大小,继而判断所需的 Virtual server 的数量,而上述这几项都是比较难以预测的。而且,每个物理节点要维护它所有的虚拟节点在系统中的位置信息,导致所需存储空间的增大,并且由于节点的增多使得之间通信量增加,从而增大了链路带宽使用,因此该方案实现较为困难。

对地址空间均衡的方案实际上是一种负载均衡的前置解决方案,在系统建立时就根据节点负载进行节点的位置安排,而 P2P 系统本身节点的加入、离开动作是很频繁的,这样每个节点的地址空间实际上是在不断地改变,从而每个节点的负载量也不在断地变化,因此,这种前置的方案对负载均衡的调节是很有限的。另外,前置的方案必须假定资源大小相同并且负载是均匀分布在系统内的地址空间上的,而实际情况并非如此,节点可能由于一个巨大资源的到来而处于超载状态,即使它只占有一个标识符空间。因此,本文在考虑系统内节点异构性及负载分布不均衡性的前提下,提出在系统的运作过程中动态地依据节点负载平滑度进行调整的负载均衡算法。

2 方案描述

本文假定资源标识符是依据统一的 HASH 函数计算得到的,资源数量要远大于系统内节点的数量,每个资源大小不同并且资源标识符的分布是不均匀的。这样,占有相同标识符空间的节点并不一定存储相同数量或相同资源负载量的数据,现有的方案没有考虑这种情况下的负载均衡问题。这里

*)基金项目:国家重点基础研究发展计划("973"计划)(2007CB307102)。于 婧 博士研究生,主要研究方向为对等网络体系结构及路由;张建辉 博士研究生,主要研究方向为 IP 网络路由协议;汪斌强 教授,博士生导师,主要研究方向为宽带 IP 网络。

对异构性的考虑只针对节点的负载，即各节点存储容量不同。

2.1 符号描述及定义

假定节点 n 的前节点(predecessor)节点标识符为 predecessor (n)，记节点 n 的地址空间为 (predecessor (n), n]，地址空间长度 $d_n=|n-\text{predecessor}(n)|$。

定义 1 节点 n 的负载容量为 C_n，当前负载量为 L_n，则节点 n 的负载利用率 $\gamma_n=\frac{L_n}{C_n}$。若系统中总节点数为 N，则系统总负载容量 C，当前总负载量 L 分别为 $C=\sum_{i=0}^{N-1}C_i$，$L=\sum_{i=0}^{N-1}L_i$。系统的平均负载利用率 $\gamma=\frac{L}{C}$，要达到 P2P 系统负载均衡也就是要求 $\gamma=\gamma_i$，$\forall i\in[0,N-1]$。记 γ_{thresh} 为负载利用率的门限值，超过这个门限就认为是超载。

定义 2 记系统内节点的最大负载利用率与最小负载利用率的比值 $\lambda=\frac{\gamma_{\max}}{\gamma_{\min}}$ 为系统的负载平滑度，则可以以系统的负载平滑度来衡量系统负载均衡的程度，也就是 λ 越趋近于 1，系统负载均衡程度越高。

定义 3 记 $\rho_n=\frac{L_n}{d_n}$ 为节点 n 在它的地址空间内的负载密度。若负载是均匀分布在系统内的地址空间上，则 $\rho_n=\rho_i$，$\forall i\in[0,N-1]$。

定义 4 记 L_n^k，$\forall k\in[\text{predecessor}(n)+1,n]$ 为节点 n 在它的地址空间中第 k 个标识符占用的存储空间，称 $\{L_n^k,\ k=\text{predecessor}(n)+1,\cdots,n\}$ 为节点 n 的存储分布；$\{\rho_n^k,\ k=\text{predecessor}(n)+1,\cdots,n\}$ 为节点 n 的存储分布密度，其中 $\rho_n^k=\frac{L_n^k}{L_n}$。

2.2 算法描述

文献[10]提出的 3 种虚拟节点的转移方法：一对一、一对多、多对多。在一对一模式中，非过载节点随机地选择一个节点探测，当发现被探测节点是过载节点时转移该节点的虚拟节点。在一对多模式和多对多模式中，系统中有多个目录服务节点(directories)保存节点的负载信息，由目录服务节点生成负载转移策略。文献[11]扩展了一对多和多对多模式，使算法适应动态 P2P 系统。仿真实验表明，算法可以解决动态系统下的负载均衡问题，但是负载均衡算法完全依赖于目录服务节点，这种类似于集中式的算法将产生单点失败问题。另外，上述虚拟节点的转移方法没有考虑转移节点间的距离问题，这种负载转移方式导致查询时产生较大的时延。因此，超载节点在选择节点进行负载转移时要尽可能地选择距离自己近的节点，假定该结构化 P2P 网络是符合 physical proximity[12] 的，也就是说物理位置相近的节点在标识符空间也是邻居节点，这样本文中节点负载转移限制在标识符空间的邻居节点进行。

下面以节点超载、资源加入、节点加入及离开四种操作分别进行详细的算法描述。

算法 1 负载均衡算法 1

假设 P 是当前节点；

算法描述：

1. P 计算本节点的负载利用率 $\gamma_P=\frac{L_P}{C_P}$；

2. If ($\gamma_P>\gamma_{thresh}$)

3. P 查找节点邻居表，联系本节点所有邻居节点 $\{N_i,\ i=0,\cdots,\text{number of neighbors}-1\}$；

4. 获取所有邻居节点的负载容量 C_{N_I}，当前负载量 L_{N_I}，计算该节点负载利用率 γ_{N_I}；

5. 找到负载利用率最小的节点，假定为 N_q；

6. P 计算转移到节点 N_q 上的负载大小 L_P^{trans}，使得 $\gamma_P\approx\gamma_{N_q}$ 并且保证 $\gamma_{N_q}<\gamma_{thresh}$，计算公式为 $L_P^{trans}\approx\frac{C_{N_q}L_P-C_PL_{N_q}}{C_P+C_{N_q}}$ 且要求 $L_P^{trans}<\gamma_{thresh}C_{N_q}-L_{N_q}$；

7. P 计算本节点的存储分布 $\{L_P^k,\ k=\text{predecessor}(P)+1,\cdots,P\}$，依据 L_P^{trans} 的大小将最少量的标识符对应的负载转移到节点 N_q 上；

8. P 维护转移负载对应的 index，便于查找；

9. End if

算法 1 是在超载节点的邻居节点中选择负载利用率最低的节点进行负载的转移来达到负载均衡的效果。而在算法 1 中，节点只有在超载的情况下才进行负载均衡，文献[10,11]中的负载均衡算法也是如此。若系统中所有节点都没有超载，则使用算法 1 达不到整个系统负载均衡的作用。由此提出了负载均衡算法 2，在该算法中，节点和它的邻居节点作为整个系统的一个子系统进行子系统内的负载均衡，从而逐渐达到整个系统的负载均衡的效果。

算法 2 负载均衡算法 2

输入：新的资源到来；

算法描述：

1. 计算资源依据统一的 HASH 函数计算得到资源标识符 F，记新资源为 F_{new}；

2. 按照 DHT 机制得到 F 应存贮的节点位置，假定该节点标识符为 P；

3. P 获取资源标识符为 F 的资源实际存储的位置，假定为 Q(Q 可能等于 P 也可能是 P 索引指向的节点)；

4. 获取资源标识符为 F 的资源的总负载量，假定为 L_Q^F，计算 F_{new} 到来后的总负载 $L_Q^{F'}$；

5. P 获取所有邻居节点的负载利用率 γ_{N_I} ($i=0,\cdots,\text{number of neighbors}-1$)，其中 $\gamma_Q=\frac{L_Q-L_Q^F}{C_Q}$；

6. 计算节点 P 及邻居节点存储 F 之后的负载利用率 γ'_P，γ'_{N_I}，其中 $\gamma'_{N_i}=\frac{L_{N_i}+L_Q^{F'}}{C_{N_i}}$，确定节点 M，使得 $\lambda=\frac{\gamma_{\max}}{\gamma_{\min}}$ 最小；

7. 将所有资源标识符为 F 的资源存储在节点 M 上。

算法 2 的中心思想是每次在新资源加入系统时进行该资源标识符对应的所有资源的位置的调整，使得在节点 P 的邻居节点集合中，降低 $\gamma_{\max}$ 与 $\gamma_{\min}$ 的差值，使负载平滑度 $\lambda=\frac{\gamma_{\max}}{\gamma_{\min}}$ 保持最小。算法 3 和算法 4 分别描述了节点的加入和离开对应的资源转移的操作。

算法 3 负载均衡算法 3

输入：新的节点到来，假定节点标识符为 Q；

算法描述：

1. 找到节点 Q 在系统中的位置，获取 Q 的后继节点(假定为 P)的存储分布 $\{L_P^k,\ k=\text{predecessor}(P)+1,\cdots,P\}$；

2. 根据 P 的存储分布计算得到节点标识符 Q'，使得 $\left|\frac{\sum_{i=\text{predecessor}(P)+1}^{Q'-1}L_P^i}{C_{Q'}}-\frac{\sum_{i=Q'}^{P}L_P^i}{C_P}\right|$ 达到最小值；

3. 将新加入节点的节点标识符修改为 Q' 并加入系统。

以往提出的节点加入算法大多是通过随机探测[13]的方式找到地址空间最大的节点并在它之前插入，而在资源分布不均匀的情况下，这种方式并不能保持负载均衡。算法 3 提出的节点加入算法，使得加入的节点与它的后继节点保持大

致相同的负载利用率，提高负载平滑度。

算法 4 负载均衡算法 4

输入：节点 Q 离开系统；

算法描述：

1. 获取 Q 的后继节点(假定为 P)；

2. 计算 P 在接管 Q 的负载后的负载利用率 $\gamma_P = \frac{L_P + L_Q}{C_P}$；

3. If ($\gamma_P > \gamma_{thresh}$)

4. 计算节点 P 在接管 Q 的负载后的存储分布$\{L_P^k, k=\text{predecessor}(Q)+1,\cdots, P\}$；

5. 按照新的存储分布执行负载均衡算法 1，注意这里 P 查找节点邻居表，联系本节点除节点 Q 外的所有邻居节点$\{N_i, i=0,\cdots,\text{number of neighbors}-2\}$；

6. 进行对应的负载转移；

7. Else

8. 节点 Q 将所有负载转移到 P 上；

9. End if

10. If (Q 为其他节点存有资源)

11. 将该资源的新的位置通知原存储节点；

12. 原存储节点更新对应的 index；

13. End if

14. 节点 Q 离开系统。

3 算法分析及仿真

本文提出的负载均衡算法的中心思想是在节点的邻居节点集合范围内进行基于资源标识符存储单位的负载均衡来提高系统的负载平滑度。资源转移时是以具有相同 key 值的所有资源作为一个整体进行的。然而，由于系统中资源分布是不均匀的，当某一话题为当前热门话题，它对应的资源必然增多，而 key 值有可能相同(如关键字为 mp3)。这时，这部分资源都会集中在这个 key 的 successor 上，引起的对应节点的存储的超载，称之为 key collision hotspot 问题。上述算法中对 key 值资源的整体转移在资源量巨大的情况下就会导致另一个节点的超载，而不能从根本上解决负载均衡的问题。

节点可以通过计算负载分布密度$\{\rho_n^k, k=\text{predecessor}(n)+1,\cdots, n\}$判断该区间内是否存在 key collision hotspot。对于这种单个键值引起的过载，只能采取使用容量大的节点存储或者将资源分散转移到其他节点[14]的方法，而不能简单地使用在区间内插入节点的方式或者上述整个资源标识符下资源转移的方法降低原节点的负载。然而节点交换会带来破坏系统 physical proximity 的问题，同时有 nodeId 变化引起的风险漏洞。同一资源标识符下资源分散转移的方法与本文中负载均衡算法相似，它的缺点是需要多个索引进行资源存储位置的标示，对应的查找会产生多个查询消息并行进行，从而导致消息数的增多。所以，在不是必须进行资源分散转移的情况下，尽量按照整体的方式进行负载均衡。

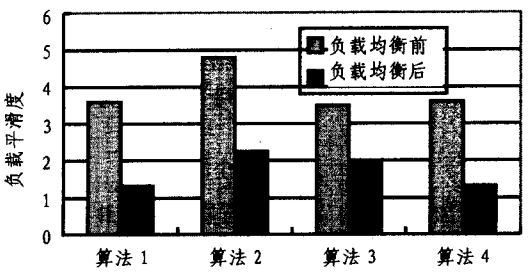

图 1 负载平滑度比较图

我们通过对 100 个节点进行仿真，节点的容量和负载量都是随机产生的，γ_{thresh} 定义为 1，仿真得到以下结果：图 1 为采用负载均衡算法前后的负载平滑度对比，可以看出负载均衡算法能有效地降低系统的负载平滑度，并且保证在节点加入、离开时的负载均衡。图 2 为进行负载均衡时系统内转移的数量与总资源数的比值，可以看到执行负载均衡算法并没有导致大量的资源转移行为，节约了系统带宽。

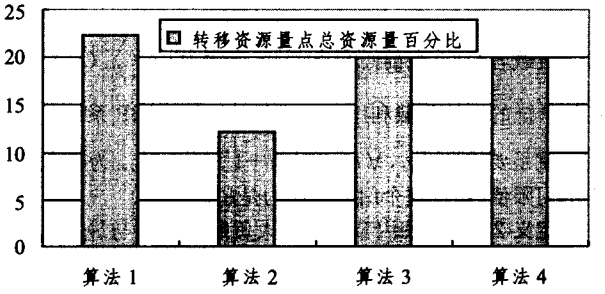

图 2 转移资源比例图

结束语 为确保异构结构化 P2P 网络中节点的负载均衡，本文提出了负载平滑度的概念及以之为衡量的负载均衡方案。方案采用基于相同资源描述符的资源整体转移方案，以节点的邻居节点为平衡范围，描述了系统在节点加入、离开，资源加入以及节点过载情况下的算法，使得整个系统逐步达到负载均衡。该方案只需进行邻居节点间的负载消息传递，并且资源转移也在距离较近的邻居节点中进行。仿真结果表明，此方案在保持较小的转移资源比例的情况下能够有效地提高系统的平滑度。

参 考 文 献

[1] Lua E K, Crowcroft J, Pias M, et al. A survey and comparison of peer-to-peer overlay network schemes. Journal of IEEE Communications Survey and Tutorial, 2005,7(2)

[2] Stoica I, Morris R, Karger D, et al. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications //Proc. ACM SIGCOMM. Aug. 2001:149-160

[3] Rowstron A, Druschel P. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems//Proc. 18th IFIP/ACM Int'l Conf. Distributed System Platforms (Middleware). Nov. 2001:329-350

[4] Zhao B Y, Kubiatowicz J D, Joseph A D. Tapestry: An Infrastructure for Fault-Tolerance Wide-Area Location and Routing. Technical Report UCB/CSD-01-1141. Computer Science Division, Univ. of California, Berkeley, Apr. 2001

[5] Ratnasamy S, Francis P, Handley M, et al. A Scalable Content-Addressable Network// Proc. ACM SIGCOMM. Aug. 2001: 161-172

[6] Karger D R, Ruhl M. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems// Proc. Third Int'l Workshop Peer-to-Peer Systems (IPTPS). Feb. 2004

[7] Saroiu S, Gummadi P K, Gribble S D. A Measurement Study of Peer-to-Peer File Sharing Systems//Proc. Multimedia Computing and Networking (MMCN), Jan. 2002

[8] Dabek F, Kaashoek F, Karger D, et al. Wide-area cooperative storage with CFS// Proc. ACM SOSP. 2001

[9] Godfrey B, Lakshminarayanan K, Surana S, et al. Load balancing in dynamic structured P2P systems//Proc. IEEE INFOCOM. Hong Kong, 2004

[10] Rao A, Lakshminarayanan K, Surana S, et al. Load balancing in structured P2P system [C]. IPTPS, Berkeleg, CA, USA, 2003

[11] Godfrey B, Lakshminarayanan K, Surana S, et al. Load balancing in dynamic structured P2P systems [C]. IEEE INFORCOM, Hong Kong, 2004

[12] Qiu T Q, Wu F, Chen G. A Generic Approach to Make Structured P2P Systems Topology-aware. Lecture Notes in Computer Science, ISPA, 2005

[13] Kenthapadi K, Manku G. Decentralized Algorithms using both Local and Random Probes for P2P Load Balancing//Proc. 17th ACM Symposium on Parallel Algorithms and Architectures (SPAA'05). Las Vegas, Nevada, USA, July 2005

[14] Gai A-T, Viennot L. Optimizing and Balancing Load in Fully Distributed P2P File. (iciw06)