一种分层P2P网络中的负载均衡算法

王 滨 沈庆国
(中国人民解放军理工大学通信工程学院 南京 210007)

摘 要 负载失衡是影响P2P系统应用服务性能的关键因素之一。目前,已有的研究集中在基于flat DHT(Distributed Hashing Table)的P2P模型上。分层拓扑结构由于其诸多优点而受到重视。将分层的思想引入虚拟服务器技术中,结合其优势,提出一种层次化的负载均衡算法。仿真实验表明,该算法可以依据节点能力的不同,保证负载在各个节点上公平分布。

关键词 P2P,层次化,负载均衡
中图法分类号 TP393 **文献标识码** A

Load Balancing Algorithm for Hierarchical P2P Networks

WANG Bin SHEN Qing-guo
(Institute of Communications Engineering, PLA University of Science & Technology, Nanjing 210007, China)

Abstract Load imbalance restricts performance of P2P applications badly. Previous researches solve P2P load imbalance focusing on flat DHT(Distributed Hashing Table) based networks. Recently, hierarchical topologies get more attention for many inherent merits. In this paper, a load balancing algorithm was proposed by merging advantages of virtual server and hierarchical technology. Simulation results show that our algorithm can ensure fair load distribution on heterogeneous nodes.

Keywords P2P, Hierarchical, Load balancing

1 引言

结构化P2P网络模型采用分布式哈希表(DHT)实现对象的存储与获取。DHT可以有效地将系统中的负载散列到各个节点上,具有一定的负载均衡能力,但相容哈希仍然会导致在规模为n的系统中节点负载的不均衡度达到$O(\log n)$级[1]。

目前关于P2P负载均衡问题的研究,可以归为如下主要几大类:节点均匀定位、节点重定位、虚拟服务器、数据项键值多维哈希、数据项重定向指针以及数据项多副本维护。这些研究主要集中在基于flat DHT的全分布式结构化P2P网络的负载均衡问题上。近年来,基于分簇的层次化拓扑结构开始逐渐受到重视。分层设计能够满足拓扑对错误的隔离和安全性要求,提供高效的缓存和带宽利用率,实现分层存储和接入控制,还可以有效缓解系统容量增大所带来的维护开销[2,3]。

本文提出了一种用于分层P2P网络的负载均衡算法。算法将虚拟服务器技术与层次化拓扑结构相结合,给出了虚拟子网和虚拟节点的概念。负载均衡过程在底层子网内进行时,节点间通过传输虚拟节点达到负载均衡的目的;若子网过载,网首间通过传输虚拟子网实现多个节点的负载均衡。算法可以依据节点能力的不同,确保负载在各个节点上的公平分布。

2 相关研究

文献[4]使用虚拟服务器VS的概念设计分布式负载均衡算法,提出了one-one,one-to-many,many-to-many 3种负载均衡机制。该算法中,维护着多个轻负载节点信息目录,每一个节点管理一个虚拟节点集,并周期性地随机与目录节点进行负载信息的交互。当节点负载过高时,将其管理的一个虚拟服务器转让给一个低负载节点,达到负载均衡的目的。文献[5]在文献[4]基础上,针对异构、动态的P2P网络设计了一种负载均衡算法。文献[6]中作者首先指出已有的P2P负载均衡算法要么没有考虑节点的异构特性,要么没有考虑拓扑一致性对负载迁移的影响。作者针对上述两个问题采用虚拟服务器的方式提出了一种有效的、位置感知的负载均衡算法。

文献[7]提出了一种支持范围查询的分层结构化P2P网络Cerco,并在其基础上分析了负载均衡。Cerco基于分层多域的原则,采用Chord算法将节点组织成多个Chord环。为了支持文件的范围查询,Cerco没有采用SHA来生成文件key,而是基于文件名的unicode码进行文件到节点的逐层映射,以保证文件按照文件名的存储有序性。Cerco的负载均衡方法采用低负载区节点移动到高负载区的策略。对于过载

到稿日期:2011-01-14 返修日期:2011-05-06 本文受混合流量下个性化适配服务系统性能研究(60472050)资助。
王 滨(1981—),男,博士生,主要研究方向为覆盖网络与对等计算,E-mail:wb_pla@163.com;沈庆国 男,教授,博士生导师,主要研究方向为下一代网络。

超级节点，可以为其增加一个子环；对于过载普通节点，只需在本环内进行负载均衡。

3 分层拓扑组织

定义 1(节点) 即参与到网络中以提供服务为目的的所有软硬件资源，它们可以是异构的。网首，即拥有至少一个子网维护权的节点，记作 s。成员节点，即子网中不拥有子网维护权的节点，记作 m。

定义 2(i 级子网) 即由 i 级网首维护的子网。i 级子网网首是某个 $i-1$ 级子网中的节点，其中 $i\geqslant 1$。顶层子网，也即 0 级子网，是由处于网络拓扑中最高层的节点构成的子网。它处在网络拓扑的最上层。底层子网处在网络拓扑的最下层，是由成员节点且只有成员节点和其网首构成的子网。非底层子网中的节点一定拥有子网维护权。

定义 3 如果 i 级子网存在，我们规定 $i-1$ 级子网网首同时也是 i 级子网网首。因此，网首可以拥有多个级别，其中的最低级别称为网首的原始级；除原始级外，网首拥有的其它级别称为网首的映射级。

定义 4 在网首维护的各级子网中，如果规定最低级别子网为其维护的第 0 层子网，则该网首维护的第 l 层子网是由第 0 层子网开始向下的第 l 级子网。

拓扑模型如图 1 所示。整个网络由多个子网构成，它们之间形成层次关系，每一个子网包含若干个节点。子网内的节点作为下一层某个子网的网首时，负责维护该子网的成员信息和成员变化，同时参与下一层子网的拓扑组织。子网内的拓扑结构较为灵活，可以采用全互联或多种结构化方式组织节点。

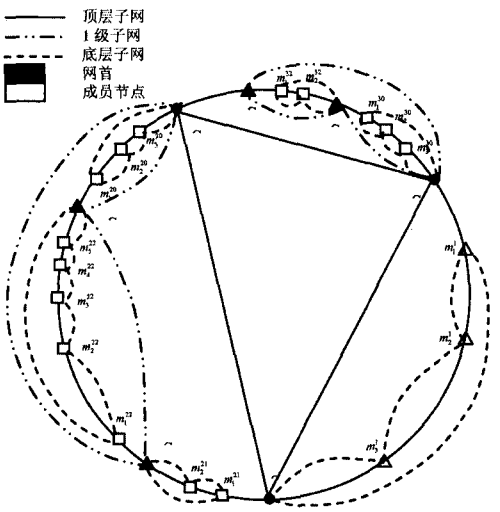

图 1 分层拓扑组织

我们对上述拓扑模型做如下形式化描述：拓扑的顶层子网记为 S，隶属于 $k-1$ 级子网 i_{k-1} 的 k 级子网 i_k 记作 $S_{i_1 i_2\cdots i_{k-1} i_k}$，其中 $i_1 i_2\cdots i_{k-1} i_k$ 是子网的逻辑标识，$1\leqslant k\leqslant L_{\max}$，$1\leqslant i_k\leqslant scale(S_{i_1 i_2\cdots i_{k-1}})$，$L_{\max}$ 是子网的最大深度(级数)，$scale(S_{i_1 i_2\cdots i_{k-1}})$ 是 $S_{i_1 i_2\cdots i_{k-1}}$ 的有效规模，即 $S_{i_1 i_2\cdots i_{k-1}}$ 中的节点作为下一级子网网首的个数。k 级子网 $S_{i_1 i_2\cdots i_k}$ 的网首记作 $s_{i_1 i_2\cdots i_k}$，k 级子网 $S_{i_1 i_2\cdots i_k}$ 的成员节点 i_0 记作 $m_{i_0}^{i_1 i_2\cdots i_k}$。特别地，顶层子网中的成员节点记作 m_{i_0}。另外，网首 $s_{i_1 i_2\cdots i_k}$ 维护的第 l 层子网记作 $S_{i_1 i_2\cdots i_k 0_{(l)}}$，其中 $0_{(l)}$ 表示连续 l 个 0。

4 负载均衡算法

4.1 虚拟服务器

为了解决 DHT 的负载不均衡问题，Chord[1] 首次提出了虚拟服务器的概念。虚拟服务器参与 Chord 的网络组织，根据其 ID 负责维护一段指定的键值空间。因此，拥有多个虚拟服务器的物理节点同时维护多段离散的键值空间，改善了负载在节点上分布的公平性。

虚拟服务器有如下两个显著的优点：首先，一次虚拟服务器传输等价于一次节点的退出和加入过程，适用于各种 DHT 结构，这种方法简单、实用。其次，虚拟服务器方式可应用于对各类负载的均衡操作，例如存储负载、计算负载和流量负载等。

4.2 层次化的负载均衡策略

定义 5(虚拟子网) 若子网 $S_{i_1 i_2\cdots i_k}$ 具有多个子网 ID 前缀，则称 $S_{i_1 i_2\cdots i_k}$ 拥有多个虚拟子网，记作 $S_{i_1 i_2\cdots i_k}^{(j_1 j_2\cdots j_k)}$。顶层子网 S 不拥有虚拟子网。

定义 6(虚拟节点) 类似于物理节点。在 DHT 中，虚拟节点同样负责维护一段键值空间。而一个物理节点可以同时拥有多个虚拟节点，因此同时负责维护多段离散的键值空间。虚拟节点包括虚拟网首和虚拟成员节点。其中，网首 $s_{i_1 i_2\cdots i_k}$ 拥有的虚拟网首记作 $s_{i_1 i_2\cdots i_k}^{(j_1 j_2\cdots j_k)}$；成员节点 $m_{i_0}^{i_1 i_2\cdots i_k}$ 拥有的虚拟成员节点记作 $m_{i_0(j_0)}^{i_1 i_2\cdots i_k(j_1 j_2\cdots j_k)}$。

结合图 1，图 2 给出了分层虚拟服务器方式下的键值空间划分。图中，每个 1 级子网拥有多个虚拟子网。在虚拟子网 $S_2^{(1)}$ 中，由隶属于 S_2 的底层子网 S_{20}，S_{21} 和 S_{22} 所拥有的虚拟子网对其维护的键值空间进行了划分，而底层虚拟子网 $S_{21}^{(12)}$ 所维护的键值空间的划分，由其虚拟网首和虚拟成员节点完成。因此，实际的负载均衡过程可分为两个部分：底层子网内节点间的负载均衡和底层子网间多个节点的负载均衡，我们称之为子网内负载均衡和子网间负载均衡。

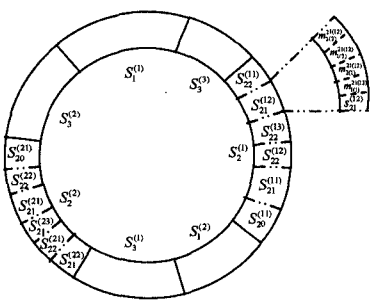

图 2 基于分层虚拟服务器方式的键值空间划分

4.3 子网内负载均衡算法

当底层子网中的节点过载时，从该子网内找到一个轻载节点进行负载均衡。底层子网中的节点会周期性地向子网内的其它节点通告自身负载情况并收集其它节点的负载信息。我们规定 $l_{m_{i_0}^{i_1 i_2\cdots i_k}}$ 表示成员节点 $m_{i_0}^{i_1 i_2\cdots i_k}$ 的负载。同样，$l_{s_{i_1 i_2\cdots i_k}}$ 表示网首 $s_{i_1 i_2\cdots i_k}$ 的负载；显然有 $l_{m_{i_0}^{i_1 i_2\cdots i_k}}=\sum_{j_1}\cdots\sum_{j_k}\sum_{j_0} l_{m_{i_0(j_0)}^{i_1 i_2\cdots i_k(j_1 j_2\cdots j_k)}}$ 以及 $l_{s_{i_1 i_2\cdots i_k}}=\sum_{j_1}\cdots\sum_{j_k} l_{s_{i_1 i_2\cdots i_k}^{(j_1 j_2\cdots j_k)}}$。$c_{m_{i_0}^{i_1 i_2\cdots i_k}}$ 表示成员节点 $m_{k_0}^{i_1 i_2\cdots i_k}$ 的能力(计算能力、存储能力、带宽资源等)。同样，$c_{s_{i_1 i_2\cdots i_k}}$ 表示网首 $s_{i_1 i_2\cdots i_k}$ 的能力。$u_{m_{i_0}^{i_1 i_2\cdots i_k}}$ 表示成员节点

$m_{k_0}^{i_1 i_2 \cdots i_k}$ 的节点利用率，$u_{s_{i_1 i_2 \cdots i_k}}$ 表示网首 $s_{i_1 i_2 \cdots i_k}$ 的节点利用率。我们规定 $u_{m_{i_0}^{i_1 i_2 \cdots i_k}} = \frac{l_{m_{i_0}^{i_1 i_2 \cdots i_k}}}{c_{m_{i_0}^{i_1 i_2 \cdots i_k}}}$，$u_{s_{i_1 i_2 \cdots i_k}} = \frac{l_{s_{i_1 i_2 \cdots i_k}}}{c_{s_{i_1 i_2 \cdots i_k}}}$。

当节点过载时，它会根据收集的负载信息找到一个轻载节点，并向该节点发送请求消息，得到确认后开始执行负载均衡。过载节点根据轻载节点的空闲可用能力信息，选择其维护的一个虚拟节点。保证执行负载均衡后，过载节点与轻载节点具有最接近的利用率。将该虚拟节点传输给轻载节点，完成均衡动作。若轻载节点无法接受过载节点所维护的任意一个虚拟节点，则过载节点根据收集的负载信息重新选择轻载节点。如果底层子网中的节点负载过高，使得底层子网的利用率高于指定阈值，则启动子网间负载均衡过程。

4.4 子网间负载均衡算法

当子网的利用率高于指定阈值 $U_{\text{threshold}}$ 时，启动子网间负载均衡算法，完成底层子网间多个节点的负载传输与均衡。这里我们规定底层子网 S_{bottom} 的负载是其网首与成员节点负载的总和，即 $l_{S_{\text{bottom}}} = l_{s_{\text{bottom}}} + \sum_{i_0} l_{m_{i_0}^{\text{bottom}}}$；底层子网 S_{bottom} 的能力是其网首与成员节点能力的总和，即 $c_{S_{\text{bottom}}} = c_{s_{\text{bottom}}} + \sum_{i_0} c_{m_{i_0}^{\text{bottom}}}$。$l_{S_{i_1 i_2 \cdots i_k}}$ 表示子网 $S_{i_1 i_2 \cdots i_k}$ 的负载，显然有 $l_{S_{i_1 i_2 \cdots i_k}} = \sum_{j_1} \cdots \sum_{j_k} l_{S_{i_1 i_2 \cdots i_k}^{(j_1 j_2 \cdots j_k)}}$，其中，$l_{S_{i_1 i_2 \cdots i_k}^{(j_1 j_2 \cdots j_k)}} = \sum_{p>k} l_{S_{i_1 i_2 \cdots i_p}^{(j_1 j_2 \cdots j_p)}}$。$c_{S_{i_1 i_2 \cdots i_k}}$ 表示子网 $S_{i_1 i_2 \cdots i_k}$ 的能力，有 $c_{S_{i_1 i_2 \cdots i_k}} = \sum_{p>k} c_{S_{i_1 i_2 \cdots i_p}}$。$u_{S_{i_1 i_2 \cdots i_k}}$ 表示子网 $S_{i_1 i_2 \cdots i_k}$ 的利用率，我们规定 $u_{S_{i_1 i_2 \cdots i_k}} = \frac{l_{S_{i_1 i_2 \cdots i_k}}}{c_{S_{i_1 i_2 \cdots i_k}}}$。

子网中的节点会周期性地向网首汇报其负载情况。网首根据收集的负载信息，计算该子网当前的负载和利用率。各级子网网首会周期性地在其所属上一级子网内通告本子网负载情况并收集其它子网的负载信息。当某个底层子网 S_{bottom} 负载度高于指定阈值 $U_{\text{threshold}}$ 时，网首 s_{bottom} 在上一级子网中根据收集的负载信息找到一个轻载子网，并向其网首发送请求消息，得到确认后开始执行负载均衡。s_{bottom} 根据轻载子网的空闲可用能力选择其维护的一个虚拟子网，以保证完成负载均衡后。过载子网与轻载子网具有最接近的负载度，将其传输给轻载子网。若 s_{bottom} 无法找到一个满足条件的轻载子网，则向上一级子网网首提出申请，负载均衡过程在更上一级子网中进行。

5 实验评价

5.1 实验环境及参数设置

我们采用德国卡尔斯鲁厄大学开发的 Inet+Omnetpp 4.0+Oversim[8] 仿真环境，验证算法的正确性和有效性。在仿真实验中，我们采用 SimpleUnderlay 模型，从 nodes_2d_15000.xml 文件中读取节点的位置信息，构造物理网络拓扑结构。

网络中的节点总数默认为 2048，形成 3 层拓扑结构。在所有节点加入网络拓扑后，负载产生器根据系统利用率、负载分布以及节点能力分布生成负载项，负载项通过关键字路由被递交到承载节点上。实验中，我们采用帕里托分布指定负载项及节点能力大小。由于其重尾特性，因此算法的性能测试是在最不利于负载均衡的情况下进行的。负载项的关键字生成采用均匀分布，使得负载项均匀分布在键值空间中。为了降低仿真实验的随机性，对每一个实验过程做 10 次重复实验，选取其中最接近平均水平的一组仿真数据作为实验结果。具体的实验参数设置如表 1 所列。

表 1 仿真环境及算法参数设置

Environment Parameter	Default value
System utilization	0.8
Object location	Uniform over key space
Object load	Pareto: shape 2, scale 0.5
Object movement cost	Equal to object load
Number of nodes	2048
Node capability	Pareto: shape 2, scale 100
Algorithm Parameter	Default value
$U_{\text{threshold}}$	0.8
layer	3
Number of id segment values in each level	4

5.2 算法性能分析

图 3(a)和图 3(b)给出了负载均衡前后节点利用率的散点分布。在算法执行过程中，当节点同时收到子网级和节点级负载均衡请求时，子网级的负载均衡请求优先于节点级负载均衡请求，上级子网的负载均衡请求优先于下级子网的负载均衡请求，这样可以避免反复、不必要的负载迁移，降低开销。可以看出，尽管在算法执行后，仍然有小部分节点处于轻量级过载状态，但负载在节点间的分布趋于公平。

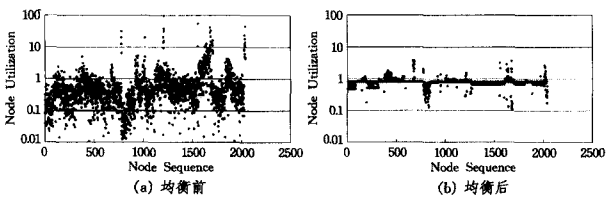

图 3 负载均衡效果

图 4 和图 5 分别给出了 id 段值取值不同的条件下第 99 个百分点的节点利用率随系统利用率和网络规模的变化。增加系统利用率会明显提高重载节点的节点利用率以及过载节点数，而算法可以将这种变化近似为线性过程。可以看出，增加分层拓扑中每一级的 id 段值个数，可以明显改善负载均衡的性能。

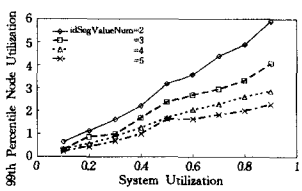

图 4 系统利用率对第 99 个百分点的节点利用率的影响

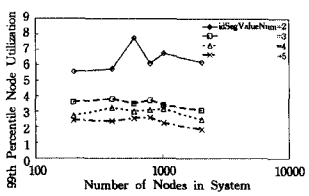

图 5 网络规模对第 99 个百分点的节点利用率的影响

结束语 引起 P2P 网络负载不均衡的原因主要有：a)节点在 ID 地址空间中的分布不均匀。b)数据项以及数据项大小分布的不均匀性。c)由于需要保证数据项的完整性，其尺寸可能有所不同，这将导致数据项占用的存储空间以及一次

(下转第 143 页)

and BookShop. L5 and Bank. M2) or(Customer. S9 and BookShop. L7 and Bank. M4) and(searchBook==0 and bookPrice==0 and balanceInquire==0 and bookID==0 and balance==0 and payInfo==0 and invoice==0 and cancel==0 and outStock==0 and payConfirm==0)

该性质验证结果为真,说明交互过程中未出现时间冲突、信息丢失等错误。

(2)安全性:"坏事情永远都不会发生"。在该实例中用户期望"Customer 在未付款的情况下收到了发货单信息 invoice"永远不会发生,其 CTL 公式表示为

A[] not(Customer. S7 and Customer. S9)

(3)活性:"好事情最终会发生"。在该实例中用户期望"在完成付款后的 10 个时间单位内收到发货单"最终会发生,其 CTL 公式表示为

A<>Customer. S8 imply Customer. S9 and x<10

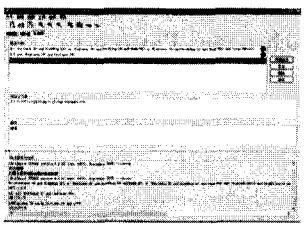

图 7 性质验证结果

结束语 Web 服务组合是面向服务计算领域的研究热点之一,而异步通信是服务组合过程中信息交互的重要特征。针对现有研究成果中较少考虑服务组合中的时间属性及异步通信问题,本文采用 XYZ/ADL 描述 Web 服务组合,将其转化为符合实时模型检测工具 UPPAAL 输入规约的 TACM 模型,利用 UPPAAL 对服务组合中的异步通信行为进行了验证。下一步我们将改进和完善时间异步通信模型 TACM,使其应用于更多的异步通信场景。另外,还将考虑如何捕捉和处理 Web 服务交互中出现的各种异常,以保证 Web 服务组合的可靠性。

参 考 文 献

[1] Pistore M, Roveri M, Busetta P. Requirements-driven Verification of Web Services[C]// Proc. of the 1st International Workshop on Web Services and Formal Methods(WSFM2004). Pisa, Italy, 2004: 95-108

[2] 雷丽晖,段振华. 一种基于扩展有限自动机验证组合 Web 服务的方法[J]. 软件学报,2007,18(12):2980-2990

[3] Foster H, et al. Model-based Verification of Web Service Compositions[C]// Proc. of the 18th IEEE International Conference on Automated Software Engineering (ASE 2003). Montreal, Canada, 2003: 152-161

[4] Fu X, Bultan T, Su J. Synchronizability of conversations among web services [J]. IEEE Transactions on Software Engineering, 2005, 31(12): 1042-1055

[5] Pallickara S, Fox G, Pallickara S L. An Analysis of Reliable Delivery Specifications for Web Services[C]// Proc. of the 10th International Conference on Information Technology: Coding and Computing(ITCC 2005). Las Vegas, USA, 2005: 360-365

[6] Vieira H T, Caires L, Seco J C. The conversation calculus: A model of service oriented computation[C]// Proc. of the 17th European Symposium on Programming(ESOP 2008). Budapest, Hungary, LNCS 4960, 2008: 269-283

[7] 何亚丽,戎玫,张广泉. 一种基于 UPPAAL 的 Web 服务组合模型检测方法[J]. 计算机科学,2010,37(11):122-125

[8] 唐稚松,等. 时序逻辑程序设计与软件工程[M]. 北京:科学出版社,2002

[9] 魏慧,戎玫,张广泉. 一种基于体系结构的 Web 服务组合描述方法[J]. 计算机工程与科学,2008,30(12):5-8

[10] Guermouche N, Godart C. Timed Model Checking Based Approach for Web Services Analysis[C]// Proc. of the 7th IEEE International Conference on Web Services (ICWS 2009). Los Angeles, USA, 2009: 213-221

(上接第 120 页)

请求需要的传输带宽资源有所不同。d)数据项具有不同的时效性,过期数据项会被节点删除,以释放资源。e)用户查询分布的不均衡,这将导致数据项具有不同的被请求频率,高请求频率会使数据项成为热点,从而增加节点的处理负载。本文针对层次化的 P2P 网络拓扑结构提出了一种负载均衡算法。仿真结果表明,本算法可以依据节点能力的不同,确保负载在各个节点上的公平分布。

参 考 文 献

[1] Stoica I, Morris R, Karger D, et al. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications[C]// Proceedings of ACM Sigcomm. Aug 2001: 149-160

[2] Lian Jie, Naik K, Agnew G B. A Framework for Evaluating the Performance of Cluster Algorithms for Hierarchical Networks [J]. IEEE/ACM Transactions on Networking, 2007, 15(6): 1478-1489

[3] Mirzaei A, Rahmati M. A Novel Hierarchical-Clustering-Combination Scheme Based on Fuzzy-Similarity Relations[J]. IEEE Transactions on Fuzzy Systems, 2010, 18(1): 27-39

[4] Rao A, Lakshminarayanan K, Surana S, et al. Load Balancing in Structured P2P Systems[J]. Lecture Notes in Computer Science, 2003, 2735: 68-79

[5] Godfrey P B, Lakshminarayanan K, Surana S, et al. Load balancing in dynamic structured P2P systems[C]// Proceedings of IEEE Infocom. Mar 2004: 2253-2262

[6] Zhu Ying-wu, Hu Yi-ming. Efficient, proximity-aware load balancing for DHT based P2P systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2005, 6(4): 349-361

[7] Rieche S, Vinh B T, Wehrle K. Range Queries and Load Balancing in a Hierarchically Structured P2P System[C]// Proceedings of 33rd IEEE Conference on Local Computer Networks (LCN). Oct 2008: 28-35

[8] Baumgart I, Heep B, Krause S. OverSim: A scalable and flexible overlay framework for simulation and real network applications [C]// Proceedings of IEEE Ninth International Conference on Peer-to-Peer Computing, Seattle. Washington, USA, 2009: 87-88