一种结构化P2P系统的负载平衡算法

魏文红[1] 向 菲[2] 王文丰[3] 王高才[4]

(东莞理工学院计算机学院 东莞 523808)[1] (河南科技大学电子信息工程学院 洛阳 471003)[2]
(南昌工程学院计算机科学与技术系 南昌 330099)[3] (广西大学计算机与电子信息学院 南宁 530004)[4]

摘 要 在结构化P2P系统中,由于使用分布式散列表,各节点能够达到基本的负载平衡。但是,各个节点由于承担负载的能力不同和数据的存取存在"热点"现象,因此仍然存在负载不平衡的情况。针对这一缺点,采用局部信息调整负载的方法,提出了分布式负载平衡算法,从而避免了单点失效问题,同时算法的实现更为简单。

关键词 分布式哈希表,P2P系统,负载平衡,虚拟服务器

中图法分类号 TP393.02 **文献标识码** A

Load Balancing Algorithm in Structure P2P Systems

WEI Wen-hong[1] XIANG Fei[2] WANG Wen-feng[3] WANG Gao-cai[4]

(School of Computer, Dongguan University of Technology, Dongguan 523808, China)[1]
(Electronic Information Engineering College, Henan University of Science and Technology, Luoyang 471003, China)[2]
(Department of Computer, Nanchang Institute of Technology, Nanchang 330099, China)[3]
(School of Computer and Electronic Information, Guangxi University, Nanning 530004, China)[4]

Abstract In DHT-based structured P2P system, as a result of the use of distributed hash table, every node can achieve the basic load balancing, but nodes have different capacity to bear the load, and data access has "flash crowds" phenomenon, so there are still load unbalancing in every node. For this disadvantage, we proposed an algorithm for distributed load balancing in which local information adjusts load. And then the problem of single node invalidation is avoided, the implement of algorithm is very simple.

Keywords DHT, Structured P2P system, Load balancing, Virtual server

结构化的P2P网络,包括Chord,CAN,Pastry,Tapestry等,在理想的情形下负载总是被均匀地分布到网络节点中。这种简单分配负载的方法从表面上看,可以达到各节点的负载平衡,但它却没有考虑节点的负载能力和数据的存取频率。即加入网络中的节点,无论是性能一般的普通PC,还是性能很好的工作站,都承担了相同的负载。这便会使得普通PC的负载已经超过了处理能力,而工作站的负载却远低于处理能力,显然这种简单地分配负载的方法是不合适的。由于节点的异构性,各个节点的带宽、存储容量、处理速度都不一样,因此更好的负载平衡算法是让各个节点各尽所能,能力越高的节点承担较大的负载。此外,在结构化的P2P网络中,一个数据对象只保存在特定的节点,即节点ID和数据ID距离最近的节点中。如果一个数据对象特别热门,那么将有更多的用户试图去获取它,这时保存该数据的节点将会处理更多的存取请求,即所谓的"热点"现象。由此,一个节点尽管保存了不多的数据,但是访问量却很大,也可能导致负载很大。

对于现有的许多负载平衡算法,大部分都是依赖网络中固定位置的某些节点,负责收集负载信息和生成转移策略。这种负载平衡算法的容错性较差,还可能出现"单点故障"的问题。还有一些负载平衡算法没有到考虑数据的存取频率,使得调整负载不能真正做到平衡。

针对目前负载平衡算法存在的这些问题,本文在基于虚拟服务器的负载平衡算法上做了更深一步的改进,提出了一种分布式负载平衡算法。它采用局部信息来调整负载,从而避免了单点失效问题,同时算法的实现更为简单;在节点的路由表上加了一个数据存取的日志,以保存数据的存取历史记录并预测将来的访问频率。

1 数据存取日志

1.1 虚拟服务器

结构化的P2P网络中,最先提出各尽所能的"负载平衡"方法的是以Chord为基础的协同文件系统CFS[5],它采用的是基于虚拟服务器的方法。每个虚拟服务器是底层DHT结构的一个"虚拟节点",负责DHT的一部分负载。加入该P2P网络的主机是"物理节点",每个物理节点负责多个虚拟节点。比如在Chord中,一个虚拟服务器负责地址空间的一部分连续区域。而一个物理节点则通过包含多个虚拟服务器来负责地址空间的几块不连续的地址空间。

到稿日期:2009-05-25 返修日期:2009-07-23 本文受国家自然科学基金(No. 60763013)和广东省科技计划项目(No. 2006B15401002)资助。
魏文红(1977—),博士,讲师,研究方向为网络与分布式计算,E-mail:hquwwh@tom.com;向 菲(1980—),女,博士,讲师,研究方向为网络信息安全;王文丰(1983—),男,博士,讲师,研究方向为智能存储系统;王高才(1976—),男,博士后,教授,研究方向为分布式网络及其算法。

把 P2P 网络上的负载分割给各个虚拟服务器，其主要优点是可以在节点之间迁移虚拟服务器来实现负载平衡。这个虚拟服务器的迁移操作，实际上就等价于先让虚拟服务器退出 P2P 网络，然后再加入 P2P 网络。退出之前，这个虚拟节点保存在负载较重的物理节点。重新加入网络后，这个虚拟节点则保存在一个负载较轻的物理节点中。通过把虚拟服务器从负载较重的物理节点迁移到负载较轻的物理节点上，就可以实现负载平衡。

尽管采用虚拟服务器的方法可能导致 DHT 内的节点增加而使路由长度变大，但由于结构化的 P2P 网络的高可扩展性，这种增加是可接受的，并且这种负载平衡的方法可用于各种基于 DHT 的结构化 P2P 网络。

1.2 数据存取日志

在节点的路由表中加入数据存取日志，用于记录数据访问的历史信息，并用这些信息来预测将来的文件访问频率，这是本文改进传统的负载平衡算法的主要思想。

文献[2]提出的负载平衡算法没有用到虚拟服务器技术，只是采用文件访问历史记录来进行负载平衡，这种方法需要为每个数据都保存访问信息，空间浪费极大。并且，由于没有采用虚拟服务器的方法，因此实现也较为复杂。

而对于大部分基于虚拟服务器的负载平衡算法，当某个节点过载时，都只是根据节点当前的负载来把虚拟服务器迁移到其它轻负载的节点。这种迁移策略只考虑了当前节点的静态负载和实际容量，没有考虑到虚拟服务器的访问频率。在这种情况下，系统没有用户以前访问数据的记录，没办法做出最佳的虚拟服务器迁移策略。更严重的是，没有数据访问记录，可能会导致虚拟服务器在各个节点间来回迁移，浪费大量的系统资源。例如，当前系统中有两个节点 A，B，节点 A 包含一个虚拟服务器 VS，不考虑虚拟服务器的访问频率。当 VS 访问频率很高时，A 超载了，则应根据一定的虚拟服务器迁移策略，将 VS 从 A 迁移到 B。在没有考虑 VS 访问频率的情况下，B 没有超载；但实际上，由于 VS 有很多用户访问，超出了 B 的处理能力，B 也超载了，则又把 VS 迁移到 A。如此反复，浪费系统资源，就无法达到真正的负载平衡。

综上所述，为了达到更好的负载平衡算法，有效地预测某个虚拟服务器将来的访问频率至关重要。在我们的算法中，为了实现这个目标，通过分析数据的历史访问记录来预测将来的访问频率。这种方法已经在 Web Cache 技术中得到证明。文献[3]证明了 Web 访问具有局部性，也就是说在过去访问非常频繁的数据，在将来仍然会以很高的概率被频繁访问。文献[4]提到，在 P2P 系统中，这种局部性特征仍然存在。

需要一个简单而有效的机制来保存数据的访问历史记录。我们知道，在 P2P 系统中，每个节点都保存了一个路由表，用于搜索资源时进行路由。每个节点都保存了当前节点的数据，这些数据是整个 P2P 网络所拥有数据的一小部分，通过完全分布式的方法由某个节点负责。同理，也可以让每个节点负责它所保存的数据的访问频率。具体到本文中，由于在进行负载迁移时，迁移的对象是一个虚拟服务器，因此只需要记录当前节点负责的虚拟服务器的访问频率就可以了。

图 1 表示当节点没有数据存取记录时节点保存的信息，它只包含了虚拟服务器及其数据。图中的节点包含 3 个虚拟服务器，虚拟服务器上保存了路由表(Route)和数据(Data)。

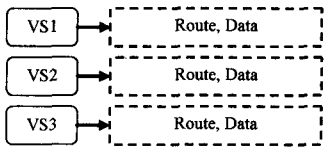

图 1 没有数据存取日志的情况 I

在图 1 中，这个节点的负载是各个虚拟服务器上的数据之和。由于这个负载是静态的，故称之为“静态负载”。但如果某个虚拟服务器被存取的频率很高，那么对带宽、处理能力、存储空间都是很大的消耗，相应的负载也变大，此时称之为“动态负载”。

显然，应该根据“动态负载”来设计负载平衡算法，而不是像文献[5]那样根据“静态负载”来进行负载平衡。

图 2 表示节点包含数据存取记录的情况。对于每个虚拟服务器，除了以前保存的数据外，还包括了一个存取记录，称之为日志(LOG)。

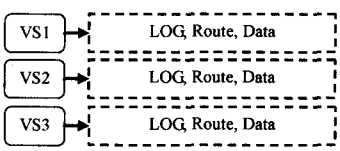

图 2 有数据存取记录的情况

图 2 中的日志数据可以根据用户的用途再定义详细的数据结构。在本文中，只需要定义两个字段：存取次数和开始统计时间。之所以需要开始统计时间，是因为数据的流行度会发生变化。比如世界杯期间，与此相关的数据在此期间都会相当热门。但过了这个时期后，又变得冷门了，这时保存的访问频率的数据就没有用处了。所以必须每隔一定的周期，刷新统计信息，重新统计。由此日志信息，可以得到节点中每个虚拟服务器的访问频率(Frequency)。

2 负载平衡算法

2.1 相关概念与定义

(1)节点的能力和负载

假设当前 P2P 网络有 N 个节点，它们的容量分别为 C_1，C_2，…，C_n，用以表示每个节点能够承受的负载的最大值。这 N 个节点的负载分别为 L_1，L_2，…，L_n。

假设当前 P2P 网络中，共有 M 个虚拟服务器。每个虚拟服务器的静态负载分别是 O_1，O_2，…，O_m；每个虚拟服务器的访问频率分别是 F_1，F_2，…，F_m。

如果一个节点 i 有 k 个虚拟服务器，则其运行时的动态负载为 $L_i=\sum_{j=1}^{k}O_jF_j$。

(2)利用率

节点的利用率 $R=L_i/C_i=\sum_{j=1}^{k}O_jF/C_i$。

对于每个节点，可以定义一个利用率超载点 R_o，比如 80%。这意味着当节点的利用率超过 R_o 后，节点对于请求的响应时间将会显著增加，即说明节点超载了，需要进行平衡负载。

(3)邻居节点

节点 A 的邻居节点(neighbors)是指 A 包含的虚拟服务器指向的物理节点的集合。

2.2 选择迁移的虚拟服务器

一个节点发现其利用率超过 R_0 时，就需要启动负载平衡算法。负载平衡算法的第一步是选择要迁移到其它节点的虚拟服务器。在选择迁移虚拟服务器时，可利用当前的访问频率来预测将来的访问频率。

因为节点的总负载为 $\sum_{j=1}^{k} O_j F_j$，其中的 $O_j F_j$ 代表一个虚拟服务器的动态负载，所以图 3 的算法中选择了动态负载最大且使得节点不再超载的虚拟服务器。如果找不到这样的虚拟服务器，则应返回动态负载最大的虚拟服务器。

```
// A:当前节点
// 返回值:要迁移的虚拟服务器
Function Choose_VS_Transfer(Node A)
{
    if(R of A<R0) // 没有超载
        return;
    retVS = nil; // 最终返回的虚拟服务器
    foreach VS in Node A do
    {
        if((A. Load-VS. O * VS. F) / A. Capability < R0)
            if(retVS == nil || retVS. O * retVS. F < VS. O *
              VS. F)
              retVS = VS;
    }
    if(retVS == nil)
        retVS equals the heaviest load VS in Node A;
    return retVS;
}
```

图 3 在过载节点 A 中选择要迁移的虚拟服务器

2.3 选择接收节点

接收节点必须满足两个条件：一是接收节点本身必须是轻载节点；二是接收节点在接收了虚拟服务器后，仍然是轻载节点。

图 4 为选择要接收虚拟服务器的节点的算法。我们采用了类似 TTL-k 泛洪来收集当前节点的邻居节点的负载信息。文献[4]表明采用这种方法是高效的，且冗余信息不会很多。文献[1]提出了基于目录节点的方法来收集节点的负载信息，但方法存在着单点失效的问题。所以，采用完全分布式的方法，不会有单点失效的问题且没有严重的冗余信息。此外，在进行泛洪迭代时，计算了邻居节点的总利用率，这样可以避免无谓的计算，加快负载平衡的算法。

```
// A:当前节点
// retVS:要迁移的虚拟服务器
// 返回值:接收节点
Function Choose_Incept_Node(Node A,VS retVS)
{
    Node incept_node;// 节点 A 的邻居节点集合
    NodeSet set = A. neighbors;k = 2;
    // 保证 set 中的节点的总利用率小于 R0
    while(k-- > 0)
    {
        foreach node in set do
            L+= node. Load,C = node. Capability;
        if(L/C < R0)
            break;
        set = set. nodes. neighbors;// 继续查找
    }
    // 求接收节点。接收节点为节点接收了虚拟服务器后利用率最
      小的节点。
    Rate rate = Ro,temp;
    foreach node in set do
    {
        temp = (node. Load + VS. O * VS. F) / node. capability
        if (temp < rate)
            incept_node = node,rate = temp;
    }
    return incept_node;
}
```

图 4 选择要接收虚拟服务器的节点

2.4 负载平衡算法

图 5 为完整的负载平衡算法。如果节点超载，那么它首先选择一个虚拟服务器，然后将之迁移到一个适合接收到该虚拟服务器的节点中。接收虚拟服务器的过程很简单，相当于虚拟服务器先退出再加入网络。

```
Function Load_Balancing(Node A)
{
    // 当前节点没有超载,直接返回
    if(A. R < R0)
        return;
    // 选择转移的虚拟服务器和接收节点
    VS transferVS = Choose_VS_Transfer(A);
    Node incept_node = Choose_Incept_Node(A,transferVS);
    if(transferVS ! = nil && incept_node ! = nil)
        transfer transferVS from A to incept_node;
}
```

图 5 完整的负载平衡算法

为了实现整个系统的负载均衡算法，节点加入 P2P 系统后，可以周期性地执行负载平衡算法。一旦发现当前节点超载了，就执行图 5 的负载平衡算法。如果用图 4 的算法没有得到适合的节点，可以借鉴文献[1]的思想，随机在网络中选择一个节点来接收虚拟服务器。

3 仿真结果及分析

在本节，通过 P2PSim 进行模拟实验，来评估第 2 节提出的负载平衡算法，并与其它的负载平衡算法进行比较。采用了 3 个不同类型的负载平衡算法的结果进行比较，分别是不采用任何负载平衡算法的 P2P 网络，称之为 NLB；采用负载平衡算法但是迁移虚拟服务器不考虑访问频率的 P2P 网络，称之为 LBNF；采用负载平衡算法也考虑访问频率的 P2P 网络，称之为 LBF。

定义超载率 $R_0 = 80\%$。节点利用率超过 R_0 即视为超载。我们称系统中超载的节点和系统中的总节点的比例为系统超载率(SOR)，用它作为负载平衡算法的评价指标。为了让模拟的结果与实际情况更符合，需要模拟实际 P2P 网络的两个特征：①网络中的节点，也就是计算机，它们的负载能力有 3 到 5 个数量级的差别[6]；②网络中 60%的查询是对 10%的数据进行的[4]。所以，在实验中，对节点能力的初始值和对

数据查询的概率都必须满足以上两个特征。此外，通过比较实际的 eDonkey 网络，在模拟实验中，文件与虚拟服务器的数目约为 1∶200，虚拟服务器与物节点的比例约为 20∶1。

因为本文提出的算法适用于任何基于 DHT 的结构化 P2P 网络，为了实现的方便，我们在实验中使用 Chord 协议在 P2PSim 上建立了 P2P 网络，并分别运行 3 个负载平衡算法。对于 Chord 网络，通过在网络运行中的不同时间点，判断网络中超载的节点个数，取平均值，求得该网络的平均系统超载率。然后扩大 Chord 网络的规模，求得网络规模改变后 3 个负载平衡算法的效果，从而判断各种方法的可扩展性。经过模拟实验，得到图 6 的结果。

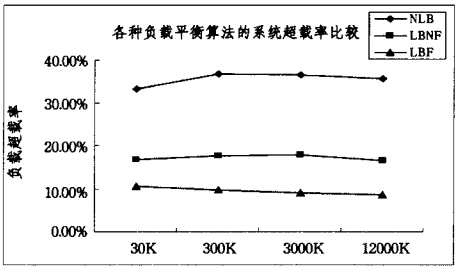

图 6　各种负载平衡算法的系统超载率比较

从图 6 可以看出，在不同规模网络运行中，分别使用这 3 种负载均衡算法，当网络规模变化时，系统超载率变化不大。说明这 3 种方法的可扩展性都很好。另外，我们提出的负载平衡算法，系统的超载率是最小的。说明采用基于存取频率的负载平衡算法，效果是三者中最好的。

结束语　本文针对结构化 P2P 系统提出了一个分布式的负载平衡算法。与现有算法相比，所提算法的主要改进有两点：①在路由表中加入了一个数据访问日志，用于保存历史访问记录并预测将来的访问频率。在选择被迁移的虚拟服务器时，考虑了被迁移的虚拟服务器的访问频率。②在选择迁移节点时采用完全分布式的方法，可以提高负载平衡算法的容错性，避免“单点失效”问题。

本文所设计的负载平衡算法是基于 DHT 的结构化 P2P 网络的增强机制。今后的工作主要集中在 P2P 网络的其他增强机制，比如拓扑一致性问题、安全问题以及无结构 P2P 网络的负载平衡问题。

参 考 文 献

[1] Rao A, Lakshminarayanan K, Surana S. Load Balancing in Structured P2P System[C]// Peer-to-Peer Systems II. Berlin: Springer, 2003, 2735: 68-79

[2] Xu Zhiyong, Bhuyan Laxmi. Effective Load Balancing in P 2 P System[C]//Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid. IEEE Computer Society Press, 2006: 81-88

[3] Fan L, Cao P, Almeida J, et al. Summary Cache a Scalable Wide-area Web Cache Sharing Protocol[J]. IEEE/ACM Transaction on Networking, 2000, 8(3): 281-293

[4] Chu J, Labonte K, Levine B N. Availability and Locality Measurements of Peer-to-Peer File System[C]// The International Society for Optical Engineering. Springer Berlin. 2002, 4868: 310-321

[5] Dabek F, Kasashoek M F, Karger D. Wide-area cooperative storage with CFS[J]. Operating Systems Review (ACM), 2001, 35 (5): 202-215

[6] Saroiu S, Gummadi P K, Gribble S D. A measurement study of peer-to-peer files sharing systems[C]// The International Society for Optical Engineering. Berlin: Springer, 2002, 4673: 156-170

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

[8] 陈贵海，李振华. 对等网络：结构、应用与设计[M]. 北京：清华出版社，2007

(上接第 70 页)

秘密通信连接。

通过分析报警数据将报警聚类并归纳出泛化报警，进而根据网络和应用环境的实际状况分析出报警原因后，可以配置或增加安全组件来阻止类似恶意行为的再度发生。但并不是所有的报警原因都能得到有效处理，有的报警原因不完全在本地网络安全的控制范围内，或者修复报警原因指明的缺陷需要高昂的代价等。这些不能得到有效处理的报警原因仍将不断导致大量的报警，这些报警无疑加大了系统处理和分析的负荷，因此有必要根据报警原因对应的泛化报警构造报警数据过滤器，用于预处理报警数据，将那些属于不能有效处理的报警过滤，以提高分析效率。

结束语　由于恶意行为一般引起多次报警，且同一恶意行为可能多次发生，而这些报警有相同的报警原因，因此，本文把逻辑上相关的报警归类到同一个报警聚类中，进而归纳为泛化报警，并由它描述报警的共同特征，从而极大地减少了报警数量，有利于准确分析出网络和应用环境面临的安全威胁，以便针对这些报警原因采取应对措施。由于 λ 参数对聚类结果有重要影响，因此，λ 参数的选择还需要进一步的研究。

参 考 文 献

[1] Valdes A, Skinner K. Probabilistic alert correlation[A]// Proc. of the RAID[C]. Davis, CA, 2001: 54-68

[2] Cuppens F, Ortalo R. LAMBDA: a language to model a database for detection of attacks[A]// Proc. of the RAID[C]. Toulouse, 2000: 197-216

[3] Dain O, Cunningham R. Fusing a heterogeneous alert stream into scenarios[A]// Proc. of the ACM Workshop on Data Mining for Sec. Applications[C]. 2001: 1-13

[4] Ning P, Cui Y, Douglas R, et al. Techniques and tools for analyzing intrusion alerts[J]. ACM Trans. on Inf. and Syst. Sec., 2004, 7(2): 274-318

[5] Ning P, Cui Y, Reeves D. Analyzing intensive intrusion alert via correlation[A]// Proc. of the RAID[C]. Zurich, 2002: 74-94

[6] Ning P, Cui Y, Reeves D. Construction attack scenarios through correlation of intrusion alert[A]// Proc. of the ACM Conf. on Computer and Comm. Sec. [C]. Washington D. C, 2002: 245-254

[7] Templeton J, Levitt K. A requires / providers model for computer attacks[A]// Proc. of the New Sec. Paradigms Workshop[C]. Cork Ireland, 2000: 31-38

[8] Julisch K. Clustering intrusion detection alarms to support root cause analysis[J]. ACM Trans. on Inf. and Syst. Sec., 2003, 6 (4): 443-471

[9] Julisch K, Dacier M. Mining intrusion detection alarms for actionable knowledge[A]// Proc. of the ACM SIGKDD[C]. Edmonton, Alterba, 2002: 366-375