一种支持邻居负载感知的动态负载平衡扩散算法

余 鑫 张 斌
(解放军信息工程大学电子技术学院 郑州 450004)

摘 要 动态负载平衡是网络节点之间负载分布调整的主要手段。负载平衡中的扩散算法与其他算法相比具有各节点同步执行,迁移开销较小、不存在中心节点瓶颈等优势。对FOS (First Order Scheme) 扩散算法进行改进,提出了支持邻居负载感知的动态负载平衡扩散算法(NLA-LB)。采用了发送者启动的策略来建立迁移组,利用节点的邻居负载信息计算负载交换的影响因子,以实现对FOS算法的加速改进。实验证明,该算法有效提升了收敛速度。

关键词 动态负载平衡,扩散算法,FOS,负载感知,收敛加速
中图法分类号 TP393 **文献标识码** A

Dynamic Load Balancing Diffusion Algorithm with Neighbors Loading Awareness

YU Xin ZHANG Bin
(Electronic Technology Institute, PLA Information Engineering University, Zhengzhou 450004, China)

Abstract Dynamic load balancing is the primary means to adjust load distribution of network nodes. Compared with other load balancing algorithm, diffusion algorithm has advantage of performing synchronously, low migrating cost without center bottleneck, and so on. The paper proposed a dynamic loading balancing diffusion algorithm with neighbors loading awareness(NLA-LB) to improve FOS diffusion algorithm. To realize convergence acceleration, sender initiated policy was adopted to establish a dynamic migrating group, and then information of neighbor loading was used to calculate loading exchanging impact factor. Experiment result shows that NLA-BA's convergence speed can be improved efficiently with lesser migrating cost.

Keywords Dynamic load balancing, Diffusion algorithm, FOS, Loading awareness, Convergence accelerating

1 引言

负载平衡是分布式系统中对资源进行合理分配的有效途径,一直是分布式系统的重要组成。互联网时代的全面到来使得系统所面临的负载压力大幅度提升,P2P网络、网格、SOA、云计算等新兴事物的出现不但没有降低负载平衡的地位,反而使其变得越发的重要。资源的有效利用关乎到系统的运行效率和服务质量,通过负载平衡就可以使系统节点处于相对平衡的状态,从而可以应对以不同分布到达的负载,最终使得系统从整体上获得效率和可靠性的提升。

Cybenko[1],Boillat[2]相继提出了经典的FOS扩散算法,它通过节点与邻居之间不断的负载交换来进行负载平衡,并最终达到全局平衡。扩散算法利用邻居间的负载信息将负载迁移限定在邻居之间,负载交换的开销较全局选取迁移对象的算法有较显著的减少,同时对于网络拓扑以及外部负载的变化也具有更强的适应性,减少了控制延迟。文献[3,4]提出通过计算扩散矩阵的特征值以及全局节点最小负载来进行加速扩散,受到了全局信息量的制约。文献[5,6]采用M2LL (Most to Least Loaded)的策略通过在邻居节点中优先选择负载最高和最低的节点对来进行结对迁移。该方法在配对时需要从全局的角度进行考虑,使得每一个节点都能找到最优的邻居。

当节点数量较多的时候需要在调整负载的有效程度和产生的开销方面进行折衷,由此出现了将系统划分成几个部分进行管理的算法,比如固定成员的固定分层算法[7]和动态临时组[8-10]负载平衡算法。前者由系统预先划分区域,设定分层管理节点,管理本区域内的负载平衡,当区域内无法进行负载平衡时才进行区域间的负载平衡,但区域失衡时迁移代价较大。而后者则在组建立后进行负载平衡,在符合一定条件之后撤销组,动态临时组的方法相比固定组更加灵活。

动态划分区域的方法可以有效地控制扩散的规模,减少不必要的迁移通信量,虽然整体负载绝对平衡度会有所下降,但是因为外部负载的随机性,可以做到将负载不平衡度维持在一个设定的范围内。本文在扩散算法的基础上,分析邻居节点所处的通道上升/下降压力,提出具有一定邻居负载感知功能的动态组扩散算法来加速收敛。

2 相关定义

假设网络可以由一个无向连通图 $G=(V,E)$ 来表示,其中节点 V 代表系统中的服务器节点集合,E 代表服务器之间

到稿日期:2012-05-21 返修日期:2012-09-19
余 鑫(1987-),男,硕士生,主要研究方向为系统工程,E-mail:yuxin3124000@yahoo.com.cn;张 斌(1969-),男,博士,教授,硕士生导师,主要研究方向为网络与信息安全。

边的集合，$E\subseteq V\times V$。令 W^t 表示负载向量，其中 w_i^t 表示节点 v_i 在第 t 轮负载平衡之后的负载量，令 w_i^0 为初始负载量。

2.1 启动节点

采用发送者启动的策略，即当系统中的某一个节点的负载超过一个特定的阈值时，它将成为一个启动节点。对于阈值的设定可以是一个固定值，也可以根据当前系统的负载分布自适应地进行调整。用 $Initate(t)$ 来表示时刻 t 的启动者集合。

2.2 K层邻居

定义 $n(v_i)$ 为节点 v_i 的邻居节点集合，$n(v_i,k)$ 表示节点 v_i 的第 k 层邻居，

$$n(v_i,k)=\begin{cases}v_i, & k=0\\ n(v_i), & k=1\\ n(n(v_i,k-1))-n(v_i,k-2), & k\geqslant 2\end{cases}$$

其中0层邻居即为节点本身，1层则为邻居节点，同理 K 层邻居则为到节点最短距离为 K 的节点集合。

2.3 迁移组

将以节点 v_i 为中心的0到 R 层邻居节点集合定义为R层迁移组：$MigrateGroup(v_i,R)=\{v_j\mid v_j\in\bigcup_{k=0}^{R}n(v_i,k)\}$。假设存在两个节点 v_i,v_j 都是启动节点，即 $v_i,v_j\in Initate(t)$，则可能出现 $n(v_i,R)\cap n(v_j,R)\neq\varnothing$ 的情况。对于这类处于两个迁移组重叠范围内的节点，本文规定节点只对第一个建组请求进行响应。

3 FOS扩散算法及其分析

3.1 FOS扩散算法基本思想

经典FOS扩散算法设想节点 v_i 与其所有的邻居都同步地进行负载的平衡。为了平衡负载需要计算节点 v_i,v_j 之间的负载交换比率 α_{ij}。对于每个节点按如下的公式进行负载更新：

$$w_i^t=w_i^{t-1}+\sum_{j\in n(v_i)}\alpha_{ij}(w_j^{t-1}-w_i^{t-1})$$

对于 α_{ij} 的选取一般存在3种形式 Cybenko[1]，Boillat[2]，Xu[11]，其中 Cybenko 和 Xu 方案中 α_{ij} 需要用到全局的信息量，而 Boillat 的方案中 $\alpha_{ij}=1/(\max(d(i),d(j))+1)$，其中 $d(i)$ 为节点 v_i 的度，不需要全局的信息量，本文在后面提到的 FOS 都指采用 Boillat 交换比例的 FOS 算法。

3.2 FOS算法分析

FOS算法采用固定节点间交换比例的方法，随着节点间的负载差异而调整迁移的负载量，即只对节点的负载差异做出反应。然而，两个负载相同的邻居节点并不具有相同的邻居负载状况，在下一次迭代之后两者的负载会发生不同的变化。

以图1的拓扑结构为例对这种情况做简单的介绍。

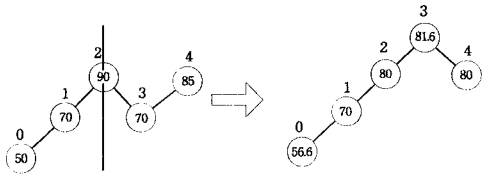

图1 FOS一次迭代结果

节点1和节点3具有相同的负载，由计算可知 $\alpha_{2,1}=\alpha_{2,3}=1/3$，所以节点2将向节点1和节点3迁移 $20\times1/3$ 大小的负载。但是，若以节点2为中心，可以发现左边的负载总量为120，右边为155，负载整体上需要从右边向左边进行迁移，但是现在的迁移量计算方式并没有很好地利用节点所处负载环境这个信息量。因此FOS算法在迁移量的计算方面存在着改进的空间，如何在不增加额外负担的前提下利用邻居负载信息对迁移进行加速是本文的主要研究内容。

4 支持邻居负载感知的动态负载平衡扩散算法

针对上述问题，利用1层邻居节点的信息可以使得对局部信息存在一定的感知能力，使得负载可以更好地向收敛方向流动。同时，利用动态分组的优势将负载调整限制在一个有限的范围内，从整体上降低不必要的负载传递。

4.1 负载交换影响因子

若节点 v_i 与节点 v_j 邻接并且有 $w_i>w_j$，则定义：

$$\delta(v_i,v_j)^t=\sum_{v_k\in n(v_j)-v_i}(w_k^{t-1}-w_j^{t-1})/d(i)*(w_i^{t-1}-w_j^{t-1})$$

将 $\delta(v_i,v_j)^t$ 称为节点 v_i 与 v_j 之间的通道压力值。

当 $\delta(v_i,v_j)^t>0$ 时，可以认为存在以下两种情况：a)节点 v_j 的邻居负载较高，所以节点上升压力也较大，故 $v_i\to v_j$ 方向也受到较大的通道压力。b)节点 v_j 的邻居中比其负载低的节点较少，且差异不大，故 $v_i\to v_j$ 方向受到的压力较大。

当 $\delta(v_i,v_j)^t<0$ 时，可以认为：a)节点 v_j 的邻居负载较低，所以节点上升压力较小，故 $v_i\to v_j$ 方向通道压力较小。b)节点 v_j 的邻居中比其负载低的节点较多，所以 $v_i\to v_j$ 受到的压力较小。

$\delta(v_i,v_j)^t$ 对节点 v_j 所处的邻居负载进行统计并与通道本身进行比较，得出通道所处的相对上升/下降压力，以此作为对原参数的改进。因为节点本身存在多个通道，故对各个通道进行归一化，得到：

$$\delta'(v_i,v_j)'=\delta(v_i,v_j)^t/\sum_{v_j\in n^-(v_i)}|\delta(v_i,v_j)^t|$$

式中，$v_j\in n^-(v_i)$ 指的是邻居节点中负载较低的节点。$\delta(v_j,v_i)'$ 作为高负载节点 v_i 与低负载节点 v_j 之间的负载交换影响因子，在进行负载的交换的时候需要结合考虑。

4.2 扩散公式改进

对扩散算法的交换比率进行改进，得到扩散公式为：

$$w_i^t=w_i^{t-1}+\sum_{j\in n(v_i)}\alpha_{ij}^t(w_j^{t-1}-w_i^{t-1})$$

式中，$\alpha_{ij}^t=\begin{cases}\dfrac{1-\delta'(v_i,v_j)'}{\max(d(i),d(j))}, & w_i>w_j\\ \dfrac{1-\delta'(v_j,v_i)'}{\max(d(i),d(j))}, & w_i<w_j\end{cases}$

α_{ij}^t 将随时间发生变化。当 $\delta'(v_i,v_j)'$ 为正时则抑制向这个方向迁移负载，$\delta'(v_i,v_j)'$ 为负则强化该方向的负载迁移。由此可以建立矩阵形式的表达式：$W^{(t+1)}=M^tW^t$，M^t 被称为系统的扩散矩阵。

若 $v_i\in MigrateGroup(v_k)\wedge v_j\in MigrateGroup(v_k)$

则 $m_{ij}^t=\begin{cases}1-\sum_{j\in n(i)}\alpha_{ij}^t, & i=j\\ 0, & j\notin n(i)\\ \alpha_{ij}^t, & i\neq j\wedge j\in n(i)\end{cases}$

若节点不是迁移组成员则与其有关的 m_{ij} 都为0，若 α_{ij}^t 不为0，则 $\alpha_{ji}^t=\alpha_{ij}^t$。

4.3 算法实现

步骤 1　初始化

本文假定系统的拓扑结构不发生改变，属于静态网络，由此需要根据系统的拓扑结构为每一个节点预先查找好 $n(v_i,k)$，将其作为节点成为启动者后构建迁移组的候选成员。可以采用经典的 floyd 算法来查找节点之间的最短距离，时间复杂度为 $O(n^2)$，与 v_i 最短距离为 $k \leqslant R$ 的节点分别归为 $n(v_i,k)$，并在本地保存。

```
floyd(G);
∀ v_i∈V,extract n(v_i,k),k∈[1,R_max]
w̄=阈值。
```

步骤 2　建立迁移组

当节点 v_i 变成启动节点时，首先向邻居节点广播一个负载信息交换包，其中带有 v_i 的已成为启动节点的信息。节点收到该包后立即向邻居节点广播自己的负载(并指出该迁移组由 v_i 发起)，收到该包的邻居节点查找自己的 K 层邻居表，若 v_i 在表中则同样广播自己的负载。当 v_i 不在节点的 K 层邻居表中或者本身已加入其他组则不广播。由此，所有可以加入迁移组的节点已经广播自己的负载，并从邻居的广播情况中可以获知邻居节点是否已经加入 v_i 的迁移组。

```
if(w_i>w̄&&v_i∉any MigrateGroup(v_k))
  send message(w_i,request(v_i,R)) to v_j∈n(v_i)
end if
for(∀ v_j∈V&& received messsage(w_k,request(v_i,R)))
   if(v_j∉any MigrateGroup(v_k)&&v_i∈n(v_j,R))
    send message(w_j,request(v_i,R)) to v_k∈n(v_j)
  end if
end for
```

步骤 3　通道压力值计算和负载交换

每一个加入迁移组的节点若存在负载比自己高的邻居节点则计算通道压力值并发送给该节点。

```
for(∀ v_j∈MigrateGroup(v_i,R))
  if(w_j<w_k)
    calculate δ^t(v_k,v_j) then send to v_k
  end if
```

节点收到通道压力值后进行归一化计算，然后交换负载。

```
if(w_k<w_j&& received δ^t(v_k,v_j))
    calculate δ^t(v_k,v_j)' --all δ received--> α_ij^t
    migrate-load(w_j,w_k)
  end if
end for
```

步骤 4　下一轮通道压力值计算和负载交换

步骤 n　终止条件：设定迁移组执行轮数，当计数减为 0 后则退出迁移组。

5　实验及结果分析

实验采用 $s^2=\frac{1}{n-1}\sum_{i=1}^{n}(w_i-\overline{w})^2$ 作为指标来表示节点负载的不平衡度，s^2 代表了各个节点与均值也就是最终收敛负载量之间的分离度。s^2 高则代表节点之间的负载差异较大，即负载的不一致程度较高。

在 MATLAB 环境下构建了两种规模的网络，其中 9 个节点的网络作为对三层迁移组的模拟，40 个节点的网络作为对五层迁移组的模拟。随后对 FOS 算法、本文提出的 NLA-LB，以及文献[5]中采用 M2ll 规则的算法进行了比较，试验结果如图 2、图 3 所示。

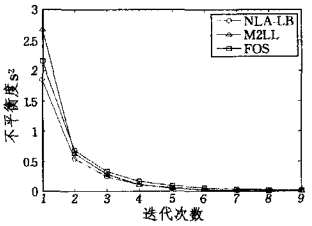

图 2　3 层 9 节点收敛结果

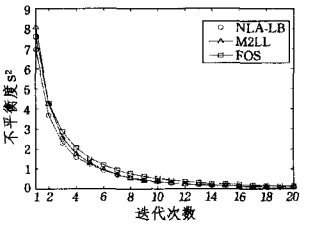

图 3　5 层 40 节点收敛结果

实验表明，本文提出的 NLA-LB 算法相比原始的 FOS 算法因为考虑了邻居的负载情况而对负载的调整更加有效。而选择所有邻居进行负载迁移的 NLA-LB 和 FOS 算法相比只选择一个邻居进行负载迁移的 M2LL 算法在前几轮的迭代中的收敛速度更快。

图 4、图 5 的结果展示了 3 种算法在不同规模下因为迁移产生的负载迁移量。

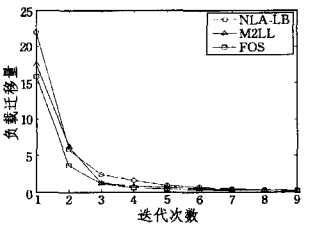

图 4　3 层 9 节点迁移量结果

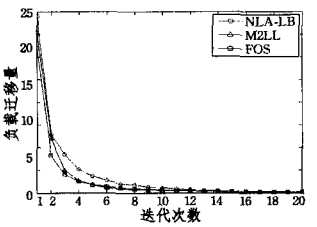

图 5　5 层 40 节点迁移量结果

由实验可知，NLA-LB 算法相比 FOS 算法需要更多的迁移量来降低整体的不平衡度，而 M2LL 算法的特点是在每轮迭代中将迁移限制在邻居对中，减少了不必要的节点间流动。较少的迁移量固然降低了通信开销，但同时也会带来收敛速度的下降，因而通过适当增大迁移量来提速也是一种合理的方法，而邻居负载感知的方法正是加大了对有效邻居节点的负载迁移量，降低了其他方向的负载迁移量，不仅实现了收敛的加速，还控制住了负载迁移量的增加规模。

结束语　本文针对局部扩散算法特点，采用发送者启动的策略，建立以过载节点为中心的 K 层迁移组，在不增加过多额外负担的条件下充分利用可获取的局部邻居信息，对节点间负载交换比例进行改进，以加速收敛。实验结果表明邻居负载感知的方法可以实现收敛的加速，同时并未带来过多的额外开销。后续将进行最优迁移组的大小选取，以及迁移组之间出现重叠成员节点对于系统整体收敛的影响等方面的研究。

参 考 文 献

[1] Cybenko G. Dynamic Load Balancing for Distributed Memory Multiprocessors[J]. Parallel Distributed Compt., 1989(7): 279-301

[2] Boillat J E. Load balancing and Poisson equation in a graph [J]. Concurr: PractExp, 1990, 2(4): 289-313

[3] Bahi J M, Couturier R, Vernier F. Accelerated Diffusion algorithms on general dynamic networks [C] // 5th International Conference, PPAM. Czestochowa Poland, 2003: 77-82

[4] Bahi J M, Couturier R, Vernier F. Synchronous distributed load balancing on dynamic networks[J]. Journal of Parallel and Distributed Computing, 2005, 65(11): 1397-1405

(下转第 196 页)

大。

表 5　软件实现的 IBAS 算法和 RSA 算法运行时间

算法	运行时间(ms)
IBAS 签名[8]	43
RSA 签名[9]	50

分析因素 b 对服务响应时间带来的影响。由 2.3 节分析可知：随着 SAML 断言传递路径长度 n 的增加，IBASPV 方案比 PKI 方案的报文传送效率提高：

$$\lambda=0.88-\frac{0.63}{n}$$

与 PKI 方案相比，IBAS 方案减少了密码运算引起的报文长度增加，从而减少了报文传输时间，进而缩短了服务响应时间，则由因素 b 带来的响应速度提升率为 λ。

综合因素 a 和 b 对响应时间的影响，可知因素 b 对基于 IBASPV 方案的服务响应速度提升率 μ 起主要作用，μ 的计算公式为：将 η_b 作为影响因子与由因素 b 带来的响应速度提升率 λ 相乘，得到：

$$\mu=\eta_b\cdot\lambda=\left(\frac{1.4}{v+1.4}\right)\left(0.88-\frac{0.63}{n}\right)$$

服务响应速度提升率 μ 与网络数据传输速率 v 和服务调用链长度 n 的关系如图 4 所示：1)当网络数据传输速率 v 固定时，随着服务调用链长度 n 的增加，IBASPV 方案比 PKI 方案有明显的性能提升；2)网络数据传输速率 v 越小，IBASPV 方案的性能提升幅度越大，随着网络传输速率 v 的增加，IBASPV 方案性能提升的幅度逐渐减小。

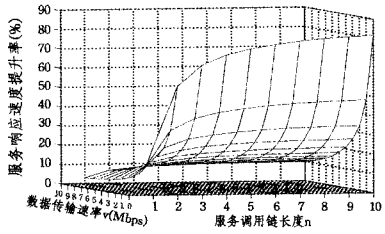

图 4　IBASPV 方案比 PKI 方案服务响应速度提升率

以下基于不同网络传输速度的 Web 服务调用环境，举例说明基于 IBASPV 协议的服务调用响应速度比基于 PKI 多次签名方案的服务调用响应速度的提升率。根据实践经验，SAML 断言传递路径长度多为 2 到 7 之间。权威机构 Akamai 在 2011 年第三季度公布的中国互联网用户终端平均网速约为 1.4Mbps，全球互联网用户终端平均网速约为 2.7Mbps[10]。据此假设，$2\leqslant n\leqslant 7$；当 $v=1.4$Mbps 时，响应速度提升约为 28%～40%；当 $v=2.7$Mbps 时，响应速度提升约为 19%～27%。若校园网环境中终端网速 $v=10$Mbps，则响应速度提升约为 7%～9%。

结束语　为了解决基于 PKI 签名机制的服务认证调用显著降低响应速度的问题，提出了基于身份聚合签名的 SAML 路径验证协议，并采用 SVO 逻辑证明了协议的安全性。该协议在保证断言传递过程中完整性、源不可伪造性和传递路径不可伪造的同时，通过减少签名后断言的增长幅度，达到提高服务响应速度的目的。通过实验分析得到了引起响应时间增加的主要因素随网络数据传输速率的变化趋势，然后对比分析了基于本文协议与基于 PKI 多次签名协议的 Web 服务认证调用性能，结果表明本文提出的 IBASPV 协议能够有效提高服务响应速度。

参 考 文 献

[1] OASIS. Assertions and Protocols for the OASIS Security Assertion Markup Language(SAML) V2.0[S]. 2005

[2] OASIS identifier: wss-v1.1-spec-os-SOAPMessage Security, Web Services Security: SOAP Message Security 1.1[S]. 2006

[3] Gentry C, Ramzan Z. Identity Based Aggregate Signatures[C]// Proc. of Public Key Cryptography. Springer, 2006

[4] Syverson P, Van Oorschot P. On Unifying Some Cryptographic Protocol Logics[C]// IEEE Computer Society Symposium on Principles of Distributed Computing. ACM Press, 1994, 5: 14-28

[5] Michael C. Pairing Calculation on Supersingular Genus 2 Curves [C]// Proc. of SAC'06. 2006

[6] Rodrigues D, Estrella J C. Analysis of security and performance aspects in service-oriented architectures [J]. International Journal of Security and its Applications, 2011, 5(1): 13-30

[7] The Apache Rampart Project[Z]. http://axis.apache.org/axis2/java/rampart, 2011

[8] Berreto. A Note on Efficient compution of cube roots in charateristic3[EB/OL]. http://eprint.jacr.org/2004/305, 2004

[9] Zhao M, Smith S, Nicol D. Aggregated Path Authentication for Efficient BGP Security[C]// ACM Conference on Computer and Communication Security. Alexandria, USA, 2005: 128-138

[10] Akamai. State of Internet report[R]. http://www.akamai.com/stateofinternetreport/, 2011

(上接第 169 页)

[5] Sider A, Couturier R. Fast load balancing with the most to least loaded policy in dynamic networks[J]. Supercomput, 2009, 49: 291-317

[6] Bahi J M, Couturier R, Sider A. Design and analysis of the M2LL policy distributed algorithm for load balancing in dynamic, networks. 2006[C]// Heidelberg, Springer, Proc of the 2006 int symp on parallel and distributed processing and applications (ISPA'06). LNCS, vol 4331, 2006: 195-204

[7] 杨夏妮，覃海生. 基于 Petri 网的动态负载平衡双层调度模型研究[J]. 广西科学院学报，2008，24(2)：296-299

[8] 王少峰，周忠，吴威. 一种面向分布式虚拟环境的分层迭代负载平衡算法[J]. 软件学报，2008，19(9)：2471-2482

[9] 王宏宇，何利娟，杜晓丽. 基于计算场的网格动态负载平衡算法[J]. 河北大学学报：自然科学版，2011，31(2)：208-213

[10] Aakanksha, Bedi P. Load balancing on dynamic network using mobile process groups[C]// 15th International Conference on Advanced Computing and Communications. 2007

[11] Xu C Z, Lau F C M. Optimal parameters for load balancing with the diffusion method in mesh networks[J]. ParallelProcess, 1994, 4(2): 139-147