第 34 卷 第 18 期 计 算 机 工 程 2008 年 9 月
Vol.34 No.18 Computer Engineering September 2008
·网络与通信· 文章编号：1000—3428(2008)18—0145—03 文献标识码：A 中图分类号：TP393.09

P2P 系统中基于副本链的一致性维护算法

苏长根，欧阳松
(中南大学信息科学与工程学院，长沙 410083)

摘 要：提出一种无结构纯 P2P 的副本一致性维护算法。利用副本节点发起的第一次更新消息在 P2P 网络中的广播，由其他收到消息的副本节点给出响应，构建副本链。副本链建立后，更新消息在副本节点间进行传播，不再在网络中洪泛。副本链的维护通过记录首次发起更新的副本节点 IP 地址完成。仿真试验证明该算法是简单有效的。

关键词：非结构化纯 P2P；副本一致性；洪泛；副本链

Replica Chain Based Consistency Maintenance Algorithm in P2P Systems

SU Chang-gen, OUYANG Song
(School of Computer Science and Engineering, Central South University, Changsha 410083)

【Abstract】 This paper presents a replica chain based consistency maintenance algorithm in unstructured P2P systems. It utilizes the broadcasting of update message to create a replica chain of a file when the update is firstly initiated by a replica node. After that update message is transferred through the replica chain without flooding. And the maintenance of the chain is performed by the broadcasted message recorded by every node receiving it. Experiments with this algorithm on the PeerSim platform show that the algorithm is effective.

【Key words】 unstructured pure P2P; replica consistency; flooding; replica chain

1 概述

近年来对等网络(Peer-to-Peer, P2P)因其成功的商业应用和潜在的技术价值已成为网络研究的热点。复制技术的使用使副本的研究成为 P2P 领域的一个重要课题。

以往的 P2P 系统主要提供静态文件的共享，文件副本的主要目的是提高搜索的效率，因此对 P2P 副本的研究主要集中在文件副本的建立策略和查找定位上。随着动态业务的发展，文件的更新比较频繁，需要设计算法维护 P2P 网络中副本资源的一致性。

副本的一致性可以分为强一致性和弱一致性。强一致性要求对数据的任意修改，都将同时作用到该数据的所有副本上，所有的副本在任何时候都保持一致；弱一致性不执行同步修改，更新消息首先传送给一个副本然后异步地传给其他副本，最终每个副本都会收到更新消息，从而达到一致状态。强一致性虽然能够保证所有副本的严格一致，但是降低了系统的并发度和扩展性。结构化的P2P网络(如Chord, CAN)可以借助分布式Hash表精确查找副本的位置，适合于维护副本的强一致性；非结构化P2P网络(如Gnutella[1])特别是纯P2P网络节点的组织没有预先指定的某种规则，维护副本的弱一致性比较现实。

目前针对P2P副本一致性研究还比较少，主要集中在非结构化P2P网络。基于广播的一致性维护算法[1-2]虽然可以让更新消息传送到网络中所有的副本，但是每次更新都让更新传遍网络中的所有节点对网络带宽是一种极大的浪费。基于概率[3-4]的更新传播方法虽然避免了更新消息在网络中的广播，但是难以保证更新消息到达所有的副本，在减少冗余信息和所有副本都得到一致性维护之间需要权衡。

2 副本链算法

传统的洪泛方法虽然可以使副本更新消息覆盖网络中的所有节点，但产生了大量冗余消息，严重地影响了网络的负载。尤其是更新次数较多时，更是对网络带宽的严重浪费。因此，要改善这种状况，必须想办法使更新消息的传播带有目的性，减少在整个网络的洪泛。采用副本链的方法正是基于这样的考虑。在继续讨论本文提出的算法之前，先做如下定义：

定义 1 副本节点：对某一文件 F，拥有该文件的 P2P 节点，简记为 RP(Replica Peer)。

定义 2 副本链：由文件标识符和拥有该文件的副本节点 IP 地址组成，简记为 RC(Replica Chain)。对每一个副本节点 RPm，需要保存一个针对该副本的副本链为

$$RCm=(FileID, \bigcup_{i=1,i\neq m}^{k} RP_i)$$

其中，$FileID$ 是对文件副本的标识符；$\bigcup_{i=1,i\neq m}^{k} RP_i$ 是除本副本节点自身外，所有包含该副本的节点地址集合；k 为副本节点个数。

因此，一旦建立了这样的副本链，更新传播算法就很简单了。当副本节点 RPm 对文件进行更新时，它向副本链中的所有节点发送更新消息：

UPDATE(UpdateMessage)
For(Every peerX in RC_m)

作者简介：苏长根(1982－)，男，硕士研究生，主研方向：网络与通信；欧阳松，教授

收稿日期：2007-10-18 **E-mail：**suchanggen1982@163.com

```
        SendMessage(peerX,UpdateMessage)
    End For
```

定义 3 副本链关键节点：第一次发起副本更新的节点，它将对副本链的维护起到关键作用，简记为 KNoRC(Key Node of Replica Chain)。

定义 4 副本链索引：由文件标识符 *FileID* 和副本链关键节点 *KNoRC* 的 IP 地址构成，用二元组表示为<*FileID*, *KNoRC*>。

虽然副本链建立后，更新传播的方法简单而高效，但是副本链的建立和维护都比较困难。因此，解决好副本链的建立和维护问题是算法能否成功的关键。

2.1 副本链的建立

如图 1 所示，假设节点 1 是第一次发起更新的节点，更新消息以洪泛的方式向网络中广播。当洪泛广播消息到达每一个节点后，节点检查自己是否有相同的副本。如果有相同副本，将给发起更新的源节点一个回馈信息。源节点收到回馈信息后将该节点加入到对应文件的副本链中。在图 1 中，节点 1、节点 3、节点 5 拥有同一个文件的副本，节点 1 发起的更新消息传到节点 3 和节点 5 后，节点 3 和节点 5 将给节点 1 发送回馈消息，如图中虚线箭头所示。节点 1 收到消息后，根据发出消息的源地址加入到文件的副本链中。等到节点 1 确认所有的副本节点都发出了反馈消息后(在这里判断更新消息是否已经传遍整个网络需要预估一个时间值，可以通过 *TTL* 值与完成一跳所用时间的乘积获得)，就将已构成的副本链向副本链中的每一个节点发送一份，如图中实线箭头所示。这样每一个副本节点都将保存一份文件的副本链。一旦文件的副本链建立起来了，更新消息就可以按照副本链算法逐一给每个副本节点发送更新消息，不再需要消息在整个网络的广播。

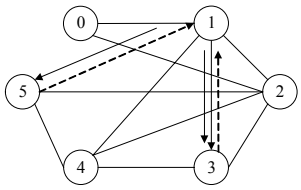

图 1 副本链的建立过程

算法过程描述如下：

更新消息的发起端：

```
Build_Replica_Chain (peer, Message)
{
Flooding(peer,Message);//以洪泛方式广播更新消息
Time timer;
List   replicaList;
While(not expire(timer))//等待收集所有回馈消息
{   If (Receive Feedback Message from peerX)
    replicaList.Add (peerX);
    Decrease(timer);
}
For( Every Peer in replicaList)
Forward(replicaList,Peer)
}
```

更新消息的接收端：

```
Process(peer ,Message)
{   If(not Receive(Message))
    {       File f=Message.File;
            If(Exist(f) In this peer)
    {   Update(f, Message.Update);
        Send(FeedBackMessage, Message.peer)
        //发送反馈消息
        }
        Store(<FileID, Message.peer>);
        //存储副本链索引
}}
```

对于不包含副本文件的 Peer 来说，它会接收到创建副本链的第一次更新消息。以往对待这样消息的方式是要么将其转发给其他节点，要么当作垃圾消息简单忽略。在本文提出的算法中，所有收到更新消息的 Peer 都将充分利用更新消息携带的信息，创建对应文件的副本链索引。在后面副本链的维护过程中将看到，这将对新副本节点加入副本链起到关键作用。

2.2 副本链的维护

由于 P2P 是一个动态网络，因此网络中的副本链也不是一成不变的。节点的加入和离开都可能影响到副本链的改变，需要对其进行维护。

2.2.1 节点的加入

一个包含某文件副本的节点加入 P2P 网络，需要加入该文件所对应的副本链，否则对副本的更新不能到达所有包含此文件的副本节点。

正如 2.1 节提到的，对于更新消息传到非副本节点，所有 Peer 节点以副本链索引的方式记录文件以及创建这个文件副本链的 *KNoRC*。如图 2 所示，当一个新的 Peer 节点 6 加入 P2P 网络时，它首先从其邻居结点 4 处获得对应的副本链索引。因为在建立副本链的过程中文件副本的更新消息曾经传到过节点 4，节点 4 保存了文件对应的 *KNoRC* 是节点 1。节点 6 将向节点 1 申请加入文件的副本链，如图中虚线所示。节点 1 将文件副本链传送给节点 6，同时告诉原副本链中其他副本节点新节点 6 的加入。节点 6 正式加入该副本链后，以后对文件的更新操作，更新消息都会传送给节点 6。

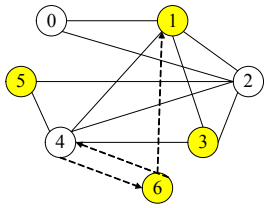

图 2 新节点加入时副本链的维护

算法过程描述如下：

```
Node_In(peer)
{       DownloadIndice(peer.Neighbours)
        For(Each file in peer)
        If(Exist a index of the file)
        { Contact(index. KNoRC)
          List replicaList;
          replicaList =Receive(index. KNoRC);
          //从 KNoRC 节点获取副本链
}}
```

2.2.2 节点的离开

节点的离开相比节点的加入稍微简单一点。对于副本链中的非 *KNoRC* 节点来说，只要向其文件对应的副本链中的每一个节点发送一条离开消息即可。同一副本链中的其他节点将从副本链中删去此节点的地址。如果退出节点是该文件的 *KNoRC* 节点，那么退出过程就不能如此简单。它需要采取如

下步骤完成对副本链的维护：

(1)在当前副本链中选择一个它认为较稳定的副本节点作为其继任者。

(2)以洪泛广播方式通知所有节点自己即将退出，选出替代自己新节点，网络中所有其他节点修改对应的副本链索引。

(3)选出的新副本节点接任 *KNoRC*，以后新加入网络的副本节点都通过新 *KNoRC* 加入副本链。

限于篇幅，这里就不再对具体算法过程进行描述。

2.3 使用轨迹标签的副本链优化算法

上述对副本链的创建和维护算法都是基于洪泛的方式进行。为了进一步提高维护效率，可以采用文献[2]提出的轨迹标签算法。用这种方式可以很好地减少洪泛在网络中冗余消息，减轻网络负载。其应用很简单，只要将上述算法中对副本链的创建和维护过程的洪泛广播改为轨迹标签算法即可。

3 性能分析

通信代价：假设 P2P 网络总节点数为 N，副本节点数为 n。副本建立之前，洪泛的方式一次传播更新的消息数为 $F(N)$ 条，副本建立之后一次更新传输的更新消息数为 $(n\text{-}1)$ 条，副本建立过程中需要传输的额外信息包括副本节点传给副本关键节点的响应消息以及副本关键节点发送给其他副本节点的副本链消息，共 $2\times(n\text{-}1)$ 条。假设消息长度都相同，并且一共执行了 m 次更新，则在不考虑节点动态加入和离开的情况下平均每次需传输的更新消息条数为

$$RCA=\{F(N)+2(n-1)+(n-1)(m-1)\}/m$$

当 m=1 时，用副本链算法与采用洪泛算法传输的消息条数差不多，甚至还多出 $2\times(n\text{-}1)$ 条用于建立副本链。但是随着副本更新次数的增多，$m\to\infty$ 时，$RCA\to(n\text{-}1)$。而采用洪泛算法，每次更新传输都接近于 $F(N)$ 条更新消息，因此平均更新数目 $FA\approx F(N)$，显然当 N 较大时，$F(N)$ 远大于 $(n\text{-}1)$。此外，如果，副本链关键节点频繁退出，也可能导致洪泛的大量产生。可以看出，本算法适合于节点较为稳定且副本更新较为频繁的 P2P 网络。

存储代价：在副本链算法中，每个副本节点都要维护一个包含其他 $(n\text{-}1)$ 个副本节点的 IP 地址。假设 *FileID* 可以用 C 个字节表示，IP 地址以 4 Byte 表示，则一个副本链字节数为 $SIZE=C+4\times(n\text{-}1)$，$n$ 个副本节点需要的总的存储空间为 $n\times SIZE$，如果再包括每个节点对副本链索引的存储，总的存储空间也不过是：$(C+4)\times N+n\times SIZE$。平均到每一个节点，所需空间也是非常少的。

对洪泛算法来说，它是不用保存有关副本的任何信息，所以副本链算法实际上是用一定数量的存储空间换取对更新消息的减少。相对于减少的更新消息的数目对网络带宽的占用，这点存储空间的代价是值得的。

4 仿真试验

为了验证副本链算法的有效性，本文采用PeerSim[5]作为仿真试验平台。PeerSim是一个用Java语言编写基于组件技术的专门用于P2P网络协议仿真的工具。通过它可以实现自己设计的算法，并对算法产生的结果进行监测和统计。

研究表明Gnutella的网络拓扑结构接近于power-law分布。本文所采用的无结构Overlay根据power-law生成。过程如下：在一个方形单元区域中，设 X_0 是中心节点，坐标为 (0.5,0.5)。该节点叫做根节点。函数 $W()$ 表示到根节点的跳数，对 i=1, 2, …, n，在方形区域中随机选择一个节点 x_i，使它与已经存在的节点 x_j 连接，并满足条件使公式 $W(x_j+\alpha\times dist(x_j,x_i)$ 的值最小。其中，$dist()$ 是一个计算欧几里德距离的函数；而 α 是一个权值参数。在本文下面的仿真试验中均采用 $\alpha=10$ 生成拓扑，以Gnutella的洪泛广播算法作为比照的对象。

图 3 显示了洪泛算法中覆盖度(覆盖度=一次更新消息传播到的副本节点数/网络中总的副本节点数)变化的趋势。网络的总节点数为 200 个，其中副本节点数为总节点数的 10%，即 20 个。从图中可以看出，随着 *TTL* 的增加，覆盖度也不断增加，这是因为，*TTL* 的增加使更新消息的传播的范围不断扩大。如图所示，*TTL*=16 时，覆盖度在 90%以上。在下面的试验过程中均采用这一 *TTL* 值。

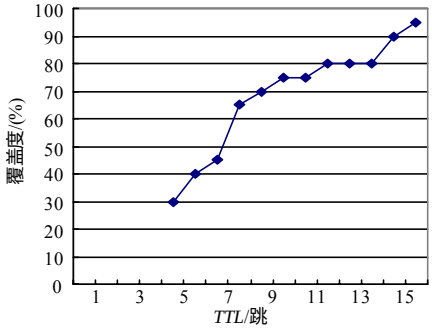

图 3 覆盖度随 *TTL* 的变化情况

由于副本链算法中，副本链的创建采用的是基于洪泛的广播算法，所以副本链算法和洪泛算法的覆盖度是一致的，因此图 3 中没有再画出相关曲线。

图 4 描述使用洪泛和副本链 2 种算法情况下，前 n 次更新传播，平均每次传输的更新消息数随更新次数的变化情况。试验中节点数仍为 200，副本节点数占总节点数的 10%，即 20 个。发起更新的副本节点是随机的。从图中可以看出，洪泛算法随着更新次数的增加，平均发送的更新消息条数始终维持在一个比较高的水平，而副本链算法除第一次更新消息和洪泛广播的相差无几，以后每次更新，平均更新消息数都会不断减少。10 次更新后，平均消息数仅为洪泛算法的 1/10 左右。这正是副本链建立极大地减少了更新消息在 P2P 网络中洪泛广播的缘故。值得注意的是，洪泛算法产生的消息数目在一个范围内波动，是因为发起更新的节点是随机指定的，不同节点发出的更新消息在网络中的路由过程是不同的。

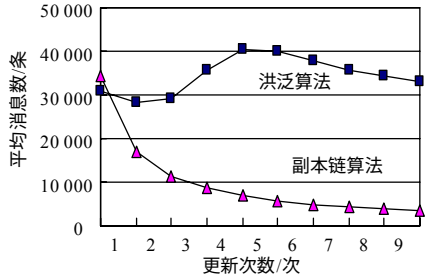

图 4 平均消息数随更新次数的变化情况

图 5 反映了洪泛和副本链 2 种算法传播的更新消息数随节点规模变化的情况。图中的横坐标代表节点数量的变化从 250 个~500 个，以 50 个递增。纵坐标为 10 次更新传输的平均消息数目，网络中副本节点的数目占网络规模的 10%。可以看到，洪泛算法受节点规模的影响明显比副本链算法要大。这是因为副本链算法在副本链建立之后，新增的传输代价仅与新增的副本节点数目有关。

(下转第 150 页)