基于 Peer-to-Peer 的分布式存储系统的设计*

徐 非+, 杨广文, 鞠大鹏

(清华大学 计算机科学与技术系,北京 100084)

Design of Distributed Storage System on Peer-to-Peer Structure

XU Fei+, YANG Guang-Wen, JU Da-Peng

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

+ Corresponding author: Phn: +86-10-51173327, E-mail: xufei97@tsinghua.org.cn, http://www.cs.tsinghua.edu.cn

Received 2002-12-26; Accepted 2003-04-15

Xu F, Yang GW, Ju DP. Design of distributed storage system on peer-to-peer structure. *Journal of Software*, 2004,15(2):268~277.

http://www.jos.org.cn/1000-9825/15/268.htm

Abstract: Distributed storage system is an important research area in peer-to-peer technology. Current research on p2p structure has made a highly controlled routing scheme with limited hops of message transfer. People now turn to pursue lower network latency that is more factual. As a storage application, distributed system must have fault tolerance-recovery capability. Based on the analysis of current research, a computing modal more approximate to real time network is constructed. A computed shortest path of nodes is used to dynamically estimate the actual latency. Adjacent nodes are gathered under an evaluating algorithm to make node latency in the same group minimal. Thus a more efficient routing can be based on node grouping. For storage persistency, an interaction management and corresponded data transfer algorithm is presented. Its locality greatly enhances the system's response to all kinds of events, and ensures the system's availability. The simulation results are provided to show that the introduction of grouping truly helps to get an effective judgment on routing choice, and can be extend to a larger scale.

Key words: distributed storage; routing scheme; grouping; fault tolerance-recovery; locality

摘 要: 分布式存储系统是p2p技术的一个重要的研究领域.当前对p2p系统的结构研究已经能够高度有效地控制节点路由次数,人们逐渐转向追求更为实际的路由距离.作为存储应用,分布式系统需要具备综合容错-恢复能力.在分析现有研究的基础上,建立一个接近实际网络节点分布的计算模型,通过已知的节点最优路径情况动态地预测网络真实路径的长度.利用评估算法聚集网络中相近的节点,使得同一分组的节点之间的距离最小化,提供更加合理的路由选择.对于存储的可靠性,提出了节点交叉管理模型和相应的数据迁移算法.这种管理策略

* Supported by the National Natural Science Foundation of China under Grant Nos.60173007, 60373004, 60373005 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2001AA111080, 2001AA104580 (国家高技术研究发展计划(863))

作者简介: 徐非(1979－),男,北京人,硕士,主要研究领域为并行/分布式系统,网格计算;杨广文(1963－),男,博士,副教授,主要研究领域为并行/分布处理,网格计算;鞠大鹏(1967－),男,博士生,讲师,主要研究领域为并行处理,高性能计算,高等计算机系统结构.

及迁移算法的本地性特点显著提高了系统对各种事件的反应能力,保证了系统的可持续性.模拟结果显示,分组为路由选择提供了确实有效的判据,而且可以扩展到更大的规模.

关键词: 分布式存储;路由方案;分组;容错-恢复;本地性

中图法分类号: TP393 **文献标识码**: A

1 问题的提出

长期以来,人们一直试图通过计算机的联合获得更强大的能力.随着对等连接的概念和有效的资源利用逐渐深入人心,p2p 的应用开始走向成熟,并和时代的趋势走到了一起.p2p 正在创造一个新的潮流——在一个系统下控制分布于世界各地的数量庞大的各种资源.

当然,这种对分散资源的整合过程并不是一件轻松的工作.对于一个无中心的分布式 p2p 系统,如何利用有限的信息从广大的资源中找到所需的结果,将成为系统性能的关键.我们在存储系统的设计中提出了基于节点分组的 p2p 系统路由模型,每个节点按照分组的不同收集和保留路由信息,并使得节点的路由选择尽量命中同一分组内的节点.以此来获得高效率的系统反应.

为了提高系统的可靠性和效率,我们在冗余之上给出一种节点交叉管理模型.每个节点上保留一部分周围节点的数据信息索引.一旦发生错误,节点能够在尽可能短的时间内,基于这个索引和数据冗余,经过简单的比较,获得新的数据存储地址,重新建立起由于错误造成的局部的节点资源的失序.

本文第 3 节将详细地介绍路由模型的设计,其中第 3.1 节给出分组的原理和依据以及初始化算法,第 3.2 节介绍分组之上的路由机制.第 4 节讨论节点的自适应和容错,并给出解决失序的数据迁移的算法.最后是对于分组的部分模拟测试结果和结论.

2 相关工作

Napster[1]是最早的实用化 p2p 系统之一,主要用来进行文件共享.它仍然是一个集中式的系统.这种集中性质为开放性的共享系统提供了强大的管理能力.而且服务器和节点之间的信息传输数据量很小,主要的数据传输在节点之间进行,也避免了网络的拥塞.另外,还有一些有代表性的共享系统,如 Freenet[2],Gnutella[3].

对于存储系统,我们更加关心数据的定位和搜索以及路由的效率.安全性也是重要的因素.显然,集中方式已经不再适用,需要一个新的体系管理系统中的数据.这方面比较成功的系统有 Oceanstore[4~6],Past[7,8]和 FreeHaven[9].Oceanstore 和 Past 都提供了一种有效的广域网存储模型.它们的底层都建立了一个代价上限为 log$N$ 的路由策略.Oceanstore 面向一个更加广义的存储概念,支持对大量复制的移动数据的串行化更新策略,但是更倾向于一个专门的服务模型,它的原型系统 Pond[6]更是描绘了一个完整的存储系统设计图景.Past 则是面向一个相对简单而紧凑的概念.它采用 Pastry[8]提供的路由机制,试图利用网络中闲置的存储节点建立一个更为完善的存储语义.p2p 的系统都强调匿名机制,但无疑 FreeHaven 在这方面做得更为出色.它建立了一个详细的匿名体系,用来防止潜在的恶意攻击.

其他一些经典的路由机制,如 Chord[10]和 CAN[11].Chord 的想法简洁而清晰,现在很多研究正在试图对它进行扩充和完善.CAN 提供了一个可无限扩充的空间网络结构,通过空间坐标和空间分割获得有效的节点路由.P2p 系统还有一些方面需要考虑,比如系统的可靠性等,有些研究[12]对此有比较详细的叙述.

3 结构设计

一个 p2p 系统必须具有良好的可扩展性,而且系统的规模往往直接关系到系统的性能.但是,在一个节点上所能够存储的其他节点信息是有限的,进行消息传递的复杂度必须是 log$N$ 的等级(N 为系统中节点的数量),才能保证路由的可行性.

在系统的底层,我们采用 DHT(distributed hash table)的路由方式,很多经典的路由算法[5,8,10,11]都基于这种方法.我们的目的主要在于如何提高每一次路由的效率,使得消息能够沿着一条在距离度量上尽量短的路径前进.

这个特点决定了我们的路由定位策略必须注重实际的网络延迟,而不再只是路由的复杂度.

虽然 Pastry[8]声称能够在很大几率上定位到距离度量最接近的节点.但是,Pastry 需要面对一个最大的难题,即它的定位算法和节点组织的基础是建立在三角不等式成立之上的,也就是在评估两个节点之间的未知距离时,认为这个距离不会超过这两个节点到第 3 个节点的已知距离之和.但是在实际网络结构中,这两个节点之间的消息传递并不一定会通过给定的第 3 个节点,所以,实际的消息传递路径的距离也不是确定的.

因此,我们希望能够有一种方法,可以确实地描述节点在网络中的分布情况.影响网络传输的因素有很多,其中地域因素就是很重要的一环.局域网和广域网之间有着数量级上的差距.这个区别决定了节点之间的距离度量随着地域分布的不同,有着不同的可估上限.而在路由和定位的过程中,为了获得较高的效率,总是希望节点能把消息尽可能快地传给下一个节点.基于这个想法,本文提出了预分组的概念.预分组的目的是给出一个节点集的划分,使得节点能够在节点空间中预测相近的节点,以此作为路由和定位的指示.

3.1 预分组

3.1.1 初始化分组算法

为了计算节点的预分组,需要有足够的已分组节点向新节点提供分组信息.系统中任何一个有效节点都可以向新节点提供信息.对于最初加入的节点,系统需要提供一个初始的节点集合 U,这些节点的分组需要事先确定好,作为整个系统的最小初始化集合.为了保证信息的数量,要求$|U|>m\cdot g$(g 为分组的数量).

初始化集合的分组显然对其后的预分组结果起着决定性的作用.预分组可以说是一种预测方法,计算所有系统节点的分布所需要的代价是我们无法承受的,所以,通过收集有限的信息并在此基础上进行推测就成为合理的选择.这种预测有时不能避免出现一些偏差,但是,如果最基本的初始分组就出现很大的偏差,再进行预测分组只会使偏差越来越大.所以,对初始集合的分组划分需要足够准确.考虑到初始集合的节点数并不多,即使计算初始节点距离矩阵代价也不是很大,可以充分地利用所有节点的信息.图 1 是初始化分组的算法.

```
Useful symbols: U-Initial Set, D_{|U|*|U|}—Distance Matrix of Initial Set, d(i,j)—The value at ith line and jth row in D, R[g]—Result sequence of grouping
(1) for (int i=0;i<g; i++){
(2) // Each loop get a grouping, total is g grouping
(3)   Get a random node n from U;
(4)   // Get n as the a grouping center
(5)   avg = Σ_{j∈U, j≠n} d(n,j) / |U|;
(6)   l = avg × adjust(g − i);
(7)   // Compute threshold as circle radius
(8)   for (; j∈U, j≠n;;){
(9)     if (d(n,j)<l){
(10)      add n to R[i];
(11) // Nodes in the circle added to grouping
(12)      U =U−{n};
(13)    }
(14) }
(15) while (U≠∅){
(16)   // For left nodes, add them to their nearest grouping
(17)   for (n∈U, int i=0; i<g; i++){
(18)     if (minSd>Sd(n,R[i])){
(19)       k = i;
(20)       minSd=Sd(n,R[i]);
(21)     }
(22)   add n to R[k];
(23)   U =U−{n};
(24) }
```

Fig.1 Initial grouping algorithm

图 1 初始化分组算法

初始集合一般节点数有限,可以并不困难地获得距离矩阵 D.算法的基本思想是依次从初始集合中随机地取出一点,以这个点为中心,一定阈值半径内的所有节点都归为一个分组.获得分组之后的节点从初始集合中移除.Avg 是当前作为中心的节点到初始集合中其他所有节点的距离的平均值.阈值半径由距离均值经过调节获得.调节函数 $adjust()$与当前的分组有关,由于经过分组初始集合中的节点数会减少,调节值应该能够及时地反映剩余节点的总体情况.反映到平面上,算法相当于用 g 个圆覆盖一个平面点集(我们认为这些圆之间互不相

交),由于圆心的随机性,可能还会有少量节点未被覆盖.对于这些节点,需要重新归组.这时候所有的分组都已经建立,我们逐一地考察剩余节点到各个分组点集之间的距离,这个距离通过剩余节点到分组内所有节点路程的标准差来衡量.节点归入最短距离的分组.

我们采用的是一个比较直观的算法.选择这个方法的原因是,它符合之后节点加入的动作,同时在模拟中可以看到这个算法也能获得足够好的结果.

衡量初始化分组的标准有两个:每个分组内节点的平均延迟和分组的节点数量.一个理想的分组结果是分组内的平均延迟近似分组的阈值半径,同时每个分组的节点数超过一个最小值.惟一能够控制这一切的就是阈值半径,所以每次初始集合的变化必然导致阈值半径的调整.

3.1.2 分组原理

预分组的基本假设是,如果两个节点到另一个节点的距离已知,则两个节点之间的距离存在一个上限.简单地说,对于节点 A,B,C,如果在网络中 A 和 B 都是 C 的近邻节点,那么 A,B 之间的距离也不会很远.也就是说,在一定的网络条件下,下面的关系成立:

$$|AC|<\varepsilon \wedge |BC|<\varepsilon \Rightarrow |AB|<\lambda(|AC|+|BC|) \tag{1}$$

其中ε为节点近邻关系判定上限,λ为放大系数.近邻关系是当两个节点之间的延迟小于一定值时所具有的关系.显然,当λ=1 时,式(1)的右端就是三角不等式关系.我们通过调节函数,要求分组的阈值半径满足近邻关系的判定.对于以节点 O 为中心的分组,落在阈值半径内的节点占绝大部分,忽略掉半径外的节点,用$\{A_n\}$表示 O 以外的节点,有以下关系:

$$Total\ Dist=\sum_{i=1}^{n}|OA_i|+\sum_{i\neq j}|A_iA_j|<\sum_{i=1}^{n}|OA_i|+\sum_{i<j}\lambda(|OA_i|+|OA_j|) \tag{2}$$

根据阈值半径小于ε,可以估计分组内的平均延迟:

$$\begin{aligned} Averange\ Dist &= 2Total\ Dist/(n+1)(n+2)\\ &< 2[\sum_{i=1}^{n}|OA_i|+\frac{\sum_{i<j}\lambda(|OA_i|+|OA_j|)]}{(n+1)(n+2)}\\ &< 2[n\varepsilon+n(n-1)\lambda\varepsilon]/(n+1)(n+2)\\ &= \frac{2n(n\lambda-\lambda+1)}{(n+1)(n+2)}\varepsilon\\ &< \frac{2n^2\lambda}{(n+1)(n+2)}\varepsilon \end{aligned} \tag{3}$$

在初始分组中平均下来,一个分组的中心外节点数 n 的值在 10~10^2 之间,分组内的最大延迟为 $2\lambda\varepsilon$,平均延迟则要略小于这个值.在网络中,λ的取值和ε有着密切的关系,当ε足够小的时候,λ的值会逐渐趋向于 1.所以,只要适当地选择ε,就可以控制λ达到我们所需要的效果.我们可以通过分组的数量来控制ε的大小.因为ε就是阈值半径的上界,所以在给定的初始集合中增加分组数量就会有效缩小各个分组的阈值半径.

我们还可以进一步地给出公式(1)的扩展形式如下:

$$|AC|<\alpha\cdot\varepsilon \wedge |BC|<\beta\cdot\varepsilon \Rightarrow |AB|<\lambda(|AC|+|BC|) \tag{4}$$

其中 $\min\{\alpha,\beta\}<\lambda<(\alpha+\beta)/2$.公式(4)试图给出这样一个关系,当两个节点中存在一条路径时,那么这两个节点之间的距离度量应该与路径的每一段距离的长度之间存在一个近似可估的关系.

3.2 路由和定位

在预分组中所做的一切,都是为了能够提高数据路由和定位的效率.为了寻找合适的存储节点或者搜索定位数据,消息必须在节点之间转发.对当前节点,如何找到一个接近自己的节点,使得消息能够快速地到达下一个节点是系统性能的关键因素.预分组的目的正在于此.

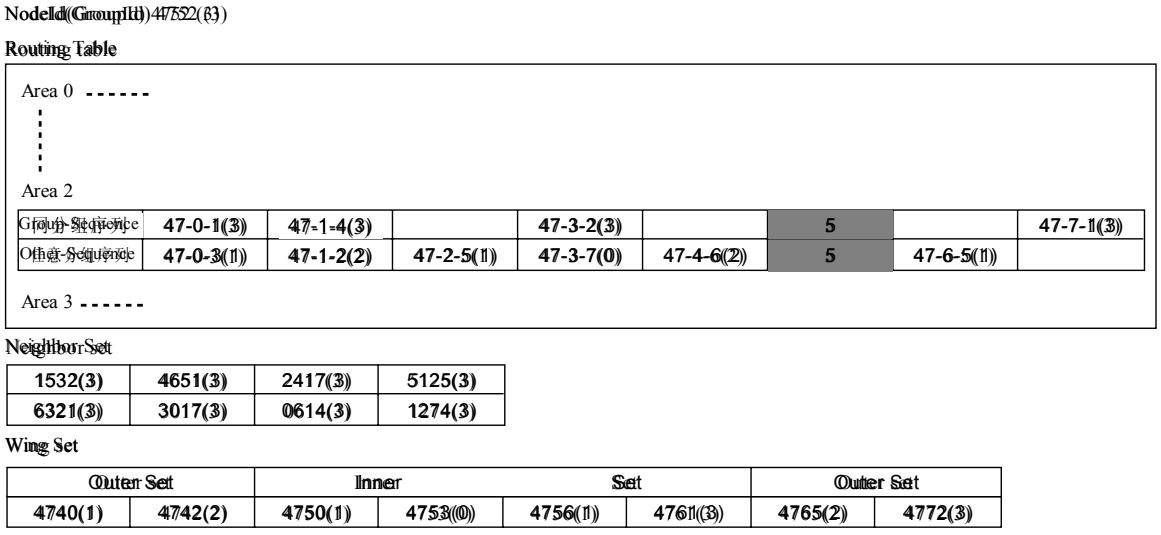
需要声明的是,预分组本身并不是严格精确的,而是建立在我们的理论上的推断.利用对已知部分网络情况的了解,我们通过预分组给出对网络真实情况的一个预测.在一个大范围的广域网络中,一个节点所能收集到的

信息与网络中的全体信息相比无疑是沧海一粟.所以,相对于确定性,我们在进行路由选择时更在意可能性.具体地说,当前节点面临众多节点的路由选择时,如果接近其中一个节点的可能性远大于其他节点,我们就选择这个节点作为路由对象.预分组反映的就是这种可能性的差异.

3.2.1 节点路由信息

为了维持节点正常的工作和路由,每个节点上都要存储一部分其他节点的信息.这些信息主要是 nodeId 到节点 IP 的映射.在路由信息的应用方式上,我们继承了 Pastry 对信息功能的划分,即路由表、邻点集和叶子集,但是其中路由表和叶子集的结构的引用模式都进行了改变,叶子集根据功能的改变改为翼集,邻点集也需要通过分组模型重新构建.

按照一般的方法,我们将 s 位的 nodeId 转变成 l 个 2^b 进制的数字序列(一般 s 的值为 128,b 的值为 4).同时,将节点的路由表划分为 l 个区域,编号为 0~l–1.在第 k 个区域中存放的 nodeId 的前 k 个数字前缀与当前节点相同,而第 k+1 位不同.每个区域中保存两个序列,其中同分组序列是和当前节点同分组的节点序列,而任意分组序列中保存其他任意分组的节点,如图 2 所示.这样做是因为对于存在 g 个分组的系统,一个分组中的节点数量基本上只有 N/g.在节点空间中寻找一个具有特定前缀的节点并不是一件有把握的事,前缀越长就越可能失败,再加上同一分组的要求,这样的节点可能根本就不存在.虽然节点具有自我完善和学习的能力,即节点具有根据每一次消息路由检查修补自己路由信息的能力,但要补全同分组序列也是一个缓慢的过程.所以对我们来说,同分组序列往往是一些片断的组合,路由的过程中需要其他的分组节点接续这些片断.



Group Sequence	47-0-1(3)	47-1-4(3)		47-3-2(3)		5		47-7-1(3)
Other Sequence	47-0-3(1)	47-1-2(2)	47-2-5(1)	47-3-7(0)	47-4-6(2)	5	47-6-5(1)	

1532(3)	4651(3)	2417(3)	5125(3)
6321(3)	3017(3)	0614(3)	1274(3)

Outer Set		Inner		Set		Outer Set	
4740(1)	4742(2)	4750(1)	4753(0)	4756(1)	4761(3)	4765(2)	4772(3)

Fig.2 Routing information on an example node. *NodeId*=4752, *GroupId*=3, and other values b=3, l=4, g=4. NodeId mapped IP is omitted

图 2 节点存储的路由信息.节点 *NodeId*=4752,分组 *GroupId*=3.相关参数 b=3,l=4,g=4. 节点 NodeId 映射的 IP 没有表示出来

邻点集则简单得多,当前节点只要收集和自己同分组的节点就可以了.邻点集中的所有节点都要满足近邻关系,所以是名副其实的邻点集合.邻点集的目的是使当前节点能够保留一些已知的距离较近的节点.

翼集中存放 nodeId 和当前节点最接近的一些节点.按照 nodeId 接近的程度,分为内集和外集,并以当前节点为中心对称分布(即 nodeId 位于当前节点两侧的节点数量相等).一般来说,内集的作用是作为冗余存储的目标节点群,所以冗余副本的数量不能超过内集的大小.外集节点作为内集的候补,一旦内集中有节点退出系统,外集中靠近内侧边缘的节点可以立刻补足内集,然后再修复外集.

3.2.2 路由策略

数据在存储系统中的移动,实际上是消息在节点之间的转发.每一个消息中都带有一个键值 D,所有的路由

选择都只是这个键值同节点的 nodeId 相互作用的结果,最终的目的在于把这个消息发送到具有和键值 D 最接近的 nodeId 的节点上去.利用 Pastry 以 nodeId 的数字位为路由转移的基本思想,应用本文提出的分组模型,我们给出了路由策略(如图 3 所示).

```
Useful symbols:
R_l^i - gs : Area l in routing table R, 0<l<⌊s/b⌋, ith
item in Group-sequence, 0<i<2^b.
R_l^i - os : Same to above, only in Other-sequence.
L_i: ith numerical cloest nodeId in wing set L.
D_l: lth digit of key D.
A: current node's nodeId.
M: neighbor set.
shl(A,B): length of share prefix between A and B.
co-g(A,B): judgment of grouping between A and B.
(1) If ( L_{-⌊|L|/2⌋} ≤ D ≤ L_{⌊|L|/2⌋} ){
(2) // D is within the range of wing set
(3)     forward to L_i, s.th. |D–L_i| is minimal;
(4) } else {      // Use the routing table
(5)     Let l =shl(D,A);        // Locate in R
(6)     if ( R_l^{D_l} - gs ≠ null )
(7)     // Hit in group-sequence
(8)         forward to R_l^{D_l} - gs ;
(9)     } else {          // Search in other-sequence
(10)        if ( R_l^{D_l} - os ≠ null )
(11)            forward to R_l^{D_l} - os ;
(12)    } else {          // Rare case
(13)        if (( ∃T ∈ L∪R∪M )&&
(14)            (co-g(T,A) = =TRUE)&&
(15)            // Prefer to search node in same grouping
(16)            ( shl(T,D) ≥ l )&&
(17)            (|T–D|<|A–D|)){
(18)                forward to T;
(19)        } else {
(20)        // Find out numerical closest NodeId
(21)            forward to  T ∈ L∪R∪M , s.th.
(22)                shl(T,D) ≥ l ,
(23)                |T–D|<|A–D|
(24)        }
(25)    }
```

Fig.3 Routing algorithm

图 3 路由算法

根据证明[7],这个算法的复杂度是 $O(\log_{2^b}N)$.从此算法与 Pastry 算法的比较可以看出,我们在算法中所增加的部分不但适应了分组的应用,而且也没有增加任何多余的消息查询等网络负载.算法在获得较快的节点路由的同时,并没有由于牺牲网络带宽而带来性能的下降.

实际操作中,为了提高效率,我们对上面的算法流程进行一些修改.确认共享前缀长度 l 之后,首先对邻点集进行搜索,寻找和键值 D 具有最大的共享前缀的节点,并且长度不小于 l.然后,如果搜索路由表的结果并不具有更长的共享前缀,就向这个节点转发.

当前节点越接近键值 D,在邻点集中找到具有更长共享前缀的节点就更加困难.但是,在路由起始的节点一端,邻点集很可能获得比路由表更好的结果,提高了路由的效率.而且路由选择的工作全部在当前节点的本地完成,先搜索邻点集的代价非常小.所以,即使是在路由的后期,仍然可以使用.

4 节点的自适应算法和容错性能

对于建立在不可信任的网络之上的 p2p 系统,必须具备两个能力:(1) 基于节点自适应学习能力的良好的可扩展性;(2) 能够及时恢复和处理由于网络的不稳定带来的错误的能力.(1)可以利用节点的自适应算法得到,而(2)则要通过冗余策略和节点之间的相互作用来完成.

4.1 节点自适应算法

节点的自适应过程主要描述的是新节点加入系统,逐步收集和完成自身的状态信息的过程.这个过程包括收集填写路由信息和计算节点的分组.新节点 X 加入系统的第 1 个步骤是获得初始的路由信息,并计算自己的分组,这需要通过 X 与一部分节点进行通信来完成.

然后,*X* 按照自己的分组调整路由表,恢复同分组序列和其他分组序列.如果 *X* 和介入节点 *A* 的分组相同,这个调整过程会非常简单;否则,调整的结果将导致 *X* 的同分组序列中的空缺比较多,不利于有效的路由.*X* 会有一个比较漫长的补完同分组序列的过程.所以,我们一般希望 *X* 加入系统时就近选择加入点 *A*,这样 *X* 有很大可能是和 *A* 同一分组,使得 *X* 能够在开始时就获得一个基本完整的同分组序列,提供高效率的路由.

4.2 容错机制

建立在大范围网络上的 p2p 系统对于加入的节点一般没有严格的控制,节点提供给系统的信息也很少.这样,节点不能向系统证明自己的可靠性,系统也无法加以区分.从整体上看,系统中的节点每时每刻都在变化着.对于存储系统,这种变化直接影响到了节点上存储数据的可用性.

4.2.1 冗余存储

冗余是通过牺牲空间来换取可靠性,对于有着广阔空间潜力的 p2p 系统是合理的选择.在每一次存储操作中,根据数据 *D* 的键值找到具有最近的 nodeId 的节点 *A*,数据首先存储到 *A*,我们把 *A* 称作 *D* 的目标存储节点.由 *A* 产生冗余所需要的 *k* 个副本,*k* 的值可以根据最近一段时期内 *A* 的翼集的变化情况来计算.然后以 *A* 为中心,对称地向 *A* 的翼集两端的节点扩散副本(要求 *k* 是偶数),存放 *D* 的副本的节点称作 *D* 的冗余节点.我们只选择 *A* 的翼集的内集作为冗余节点,所以 *k* 的最大值不能超过内集的大小.

4.2.2 节点间相互作用和数据迁移

冗余能够有效地减少由于节点意外的失效所造成的数据不可用,但是对正常的节点加入没有免疫力.而节点的加入虽然不会造成数据的丢失,但是会改变系统中节点的相对位置.相对于一个键值,新的节点取代了原有节点在系统中的位置,虽然数据仍然存放在原有节点上,但是根据键值却定位到了新节点,数据被节点屏蔽了.图 4 显示了节点变化的影响.

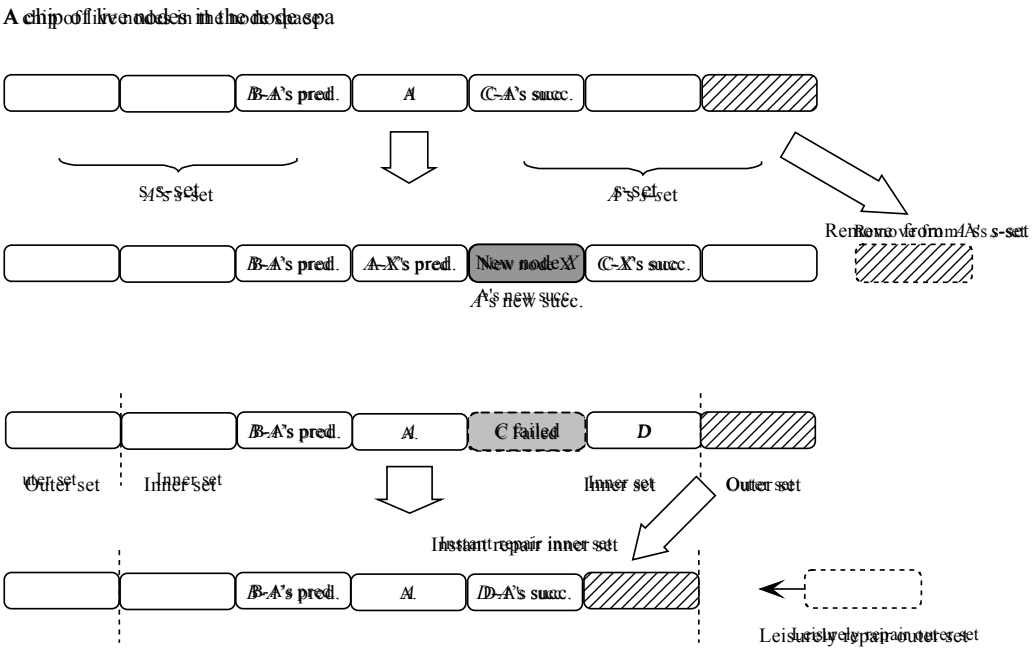


Fig.4 Changes surround node *A* and its influence

图 4 节点 *A* 周围的变化及影响

为此我们不得不考虑节点间的数据迁移.同时,为了保证迁移的效率,引入了节点间的相互控制.每一个节点保留两个指针,称为节点的前导(predecessor)和后继(successor),分别指向节点空间中位于当前节点两侧的节点.节点上还保存 3 份数据记录,分别记录本地节点以及前导和后继数据的键值.

这样对于节点 A,就有 3 个节点保存了 A 的数据信息.通过冗余策略,这些数据内容也分布在节点空间中 A 周围的节点上.如果用 A_s 表示 A 上的所有的存储数据集合,同时 A 具有前导 B 和后继 C,则有以下关系:

$$A_s = B_s \bigcup C_s .$$

也就是说,A 上的任何数据都可以在其前导节点或者后继节点上找到备份.这也是我们进行基于数据记录的数据迁移的基础.下面我们以节点加入为例给出迁移的算法,如图 5 所示.

User symbols:

A_r: Data record for node A.

D_d: Data unit of unitId D.

B=*pred*(A), B is A's predecessor.

New node X comes into B and A ($B<X<A$).

node_add_transfer()

(1) $E = A_r \bigcap B_r$; // Get the intersection of data units of A and B

(2) while ($E \neq \varnothing$){

(3) $\forall D \in E$;

(4) if (|D–A|<|D–X|){

(5) // D is on the right of X and nearer to A

(6) move D_d to X;

(7) *s_remove*(D,left); // Search left of X to get rid of the redundant data of D

(8) } else if (|D–B|<|D–X|){

(9) // D is on the left of X and more nearer to B

(10) move D_d to X;

(11) *s_remove*(D,*right*); // Search right of X to get rid of the redundant data of D

(12) } else {

(13) // D is nearest to X, X becomes the receiver for D_d.

(14) move D_d to X;

(15) if (|D–A|<|D–B|){ // Origin receiver is A

(16) *s_remove*(D,right); // Search and remove replica on the right replicate border of A

(17) } else { // Origin receiver is A

(18) *s_remove*(D,left); // Search and remove replica on the left replicate border of B

(19) }

(20) }

(21) E=E–{D};

(22) }

s_remove(D,direction)

// Search the most marginal replica on the defined direction

(1) if (direction is right){

(2) Let T =A;

(3) while(D is in T_r){

(4) T = *succ*(T);

(5) }

(6) remove D_d from T;

(7) } else {

(8) Let T =B;

(9) …// Same to above, search predecessor of T

(10) }

Fig.5 Data transferring algorithm

图 5 节点加入的数据迁移算法

在 *node_add_transfer*()中,由于 A 上拥有 A 和 B 的记录,整个过程可以由节点 A 控制完成,数据也由 A 全部传送给 X(同样也可由 B 单独完成),不需要 AB 之间的信息交换.除了进行 *s_remove*()以外,所有的信息都在本地,而进行 *s_remove*()时,A 也只需把键值 D 发送给自己的后继节点或者 B 而已.所以,算法的过程是非常高效的.

s_remove()释放边缘节点 T 上的数据,并不是把数据物理的删除,而是转移到一个 remove area 中.这样,如果发生节点的失效,T 可以立刻作为 D 的冗余节点,并且不需要再获得数据.节点每次发生数据变化时都要更新数据记录,并发送给自己的前导和后继.一般情况下,变化的内容都只是记录很小的一部分,所以只需发送有变化的部分就可以了,而对于新节点,虽然本地生成了一个新的记录,但这个记录两侧的节点可以轻松地计算得到(即 $E = A_r \bigcap B_r$),使得更新的网络代价下降到很低的水平.

对于节点的失效,也可以采用相似的做法.失效可能导致某些节点内集的损失,需要从外集补充,所以冗余不能超出内集,可以保证立刻就能找到所需要的替补节点,因为迅速地恢复失效造成的现场对我们是很重要的.至于外集的修复就可以慢慢地进行了.

另外,对于数据存储和迁移的实际操作,现在一般不倾向于对数据块不加分割的直接操作,这样的做法虽然简单、明确,但是效率不高.现在应用很多的是 Erasure code[6,13,14]的方法,但这不影响我们的讨论,所以这里不再详述.

5 分组模型模拟结果

当前的模拟主要集中在初始化分组的结果.我们使用 Transit-Stub[15]模型获得一个有 600 个节点的网络拓扑作为初始的分组集合,一共 16 个分组.通过 Dijkstra 最短路算法计算节点之间的延迟距离,根据公式(1),对不同的λ值,AB 的实际距离使用在 0.5~λ倍最短路径之间的随机值.

5.1 静态分组

图 6 是公式(1)中λ不断增大时分组的效果.分组的目的是将距离接近的节点划分到一起.从图中可见,随着λ的增大,节点之间的距离也逐渐增大.但是无论λ怎样变化,分组中节点的平均距离都要远远低于分组间的距离.节点间的距离差异已经被分组凸现出来了.

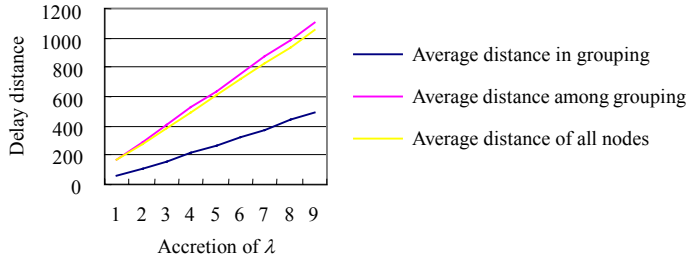


Fig.6 Static grouping result

图 6 静态分组结果

5.2 优化模型

根据公式(4),按照节点之间的距离动态地设置λ的值.图 7 是对 10 个不同的初始点集分组得到的结果.从全节点平均距离来看,不同的初始集差别很大,而分组内节点平均距离的变化却很平缓,这说明分组确实过滤掉了远距离的节点,而将互相接近的节点聚集到了一起.

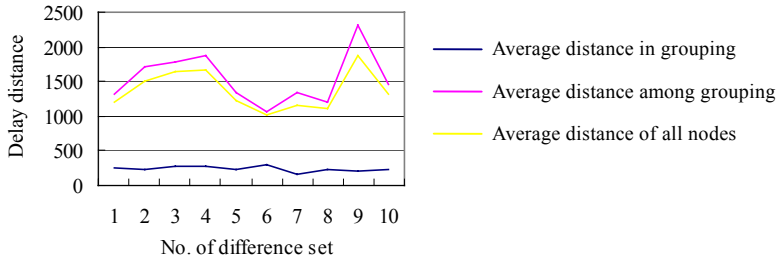


Fig.7 Dynamic optimal result

图 7 动态优化结果

这个结果对于随后的节点动态加入有着重要的意义.我们在初始化分组时使用了比较直观的点集覆盖方法.这种计算方法接近于节点加入时判断分组的计算.这部分地解释了初始集合分组结果的可推广性.

6 结 论

本文详细描述了一种建立在 p2p 上的分布式存储系统的设计思想和体系结构,给出了基于节点预分组的有效的路由策略,并提出了预分组的理论和算法.通过分组,我们给出了节点之间延迟距离的预测,并且根据这

种预测决定节点的路由选择.这样就提供了一种更加接近网络动态的预测方法,降低了网络本身的不稳定性所带来的效率损失.之后,为了维护系统的高可用性,给出了冗余策略和数据迁移算法,对系统的可靠性和稳定性提供了可靠的保证,同时提高了系统对发生错误后的恢复和应变能力.在后续工作中,我们将关注预分组中几个主要参数之间的相互影响,并作进一步的研究.

References:

[1] Napster. http://www.napster.com/

[2] Clarke I, Sandberg O, Wiley B, Hong TW. Freenet: A distributed anonymous information storage and retrieval system. In: Workshop on Design Issues in Anonymity and Unobservability. Berkeley: ICSI, 2000. 311~320.

[3] The Gnutella protocol specification v0.4 (document revision 1.2). 2001. http://www9.limewire.com/developer/gnutella protocol 0.4.pdf

[4] Kubiatowicz J, Bindel D, Chen Y, Czerwinski S, Eaton P, Geels D, Gummadi R, Rhea S, Weatherspoon H, Weimer W, Wells C, Zhao B. Oceanstore: An architecture for global-scale persistent store. In: Proc. of the ACM ASPLOS. http://www.cs.berkeley.edu/~kubitron/papers/

[5] Zhao BY, Kubiatowicz JD, Joseph AD. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report, UCB//CSD-01-1141, Berkeley, 2001.

[6] Rhea S, Eaton P, Geels D, Weatherspoon H, Zhao B, Kubiatowicz J. Pond: The OceanStore Prototype. In: Proc. the 2nd USENIX Conf. on File and Storage Technologies (FAST 2003). 2003.

[7] Druschel P, Rowstron A. PAST: A large-scale, persistent peer-to-peer storage utility. In: Proc. of the HOTOS Conf. 2001. 75~80.

[8] Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science, 2001,2218:329~350.

[9] Dingledine R, Freedman MJ, Molnar D. The free haven project: Distributed anonymous storage service. In: Proc. of the Workshop on Design Issues in Anonymity and Unobservability. 2000. 67~95.

[10] Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for Internet applications. In: Proc. of the ACM SIGCOMM. San Diego, 2001. 149~160.

[11] Ratnasamy S, Francis P, Handley M, Karp R. A scalable content-addressable network. In: Proc. of the 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications. 2000. 161~172.

[12] Bhagwan R, Savage S, Voelker GM. Understanding availability. In: Proc. of the 2nd Int'l Workshop on Peer-to-Peer Systems.

[13] Bloemer J, Kalfane M, Karp R, Karpinski M, Luby M, Zuckerman D. An XOR-based erasure-resilient coding scheme. Technical Report, TR-95-048, Berkeley: The Int'l Computer Science Institute, 1995.

[14] Zhang Z, Lian Q. Reperasure: Replication protocol using erasure-code in peer-to-peer storage network. In: Proc. of the 21st IEEE Symp. on Reliable Distributed Systems (SRDS 2002). 2002.

[15] Zegura EW, Calvert KL, Bhattacharjee S. How to modal an internetwork. In: Proc. of the INFOCOM'96. 1996. 594~602.