基于动态区间映射的数据对象布局算法*

刘 仲+, 周兴铭

(国防科学技术大学 计算机学院,湖南 长沙 410073)

A Data Object Placement Algorithm Based on Dynamic Interval Mapping

LIU Zhong+, ZHOU Xing-Ming

(School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: liuzhong@zhmail.com, http://www.nudt.edu.cn

Received 2004-09-21; Accepted 2005-05-31

Liu Z, Zhou XM. A data object placement algorithm based on dynamic interval mapping. *Journal of Software*, 2005,16(11):1886−1893. DOI: 10.1360/jos161886

Abstract: Efficient and scalable data management becomes increasingly important in large-scale distributed storage systems. A key enabling technique is a flexible, balancing and scalable data object placement and location scheme that automatically adapts to the additions or departures of storage nodes. In this paper, a data object placement algorithm based on dynamic interval mapping is proposed, which is probabilistically optimal in both distributing data evenly and minimizing data movement when storage nodes is changed. Moreover, this algorithm supports weighted allocation of the storage nodes and variable levels of the object replication.

Key words: dynamic interval mapping; data placement; object storage; balanced distribution; scalable

摘 要: 高效、可伸缩的数据管理在大规模分布存储系统中日益重要,关键是需要一种能够自动适应存储节点增加或减少的灵活、均衡和可伸缩的数据对象布局与定位方法.提出了一种基于动态区间映射的数据对象布局算法,在均衡数据分配和最少迁移数据方面都是统计意义上最优的,并且支持按照存储节点的权重分配数据和任意的数据对象副本.

关键词: 动态区间映射;数据布局;对象存储;均衡分布;可伸缩

中图法分类号: TP311 文献标识码: A

由于大量数据密集型应用的出现,如能源研究和模拟,高能物理研究,地球、海洋和大气科学研究,全球气温变化和天气预报研究,地震数据分析,大规模信号和图像处理应用研究等,建立满足数据存储容量快速增长的可伸缩大规模存储系统日益重要.可伸缩性要求使得这种大规模存储系统不可能一次性完成,它必须根据存储容量的需求,不断增加新的存储节点,同时淘汰系统中旧的存储节点.不同时期加入的存储节点权重(处理器能力、内存大小、存储容量等)是不相同的.系统必须根据存储规模的变化自动地实现存储数据的重新布局,使得存储数据能够根据存储节点权重均衡地分布到各个新的存储节点,同时保证用户对数据的访问是可靠的,即无论存

* Supported by the National Natural Science Foundation of China under Grant No.60503042 (国家自然科学基金)

作者简介: 刘仲(1971－),男,湖南邵东人,博士生,副研究员,主要研究领域为网络存储,并行与分布处理;周兴铭(1938－),男,教授,博士生导师,中国科学院院士,CCF 高级会员,主要研究领域为高性能体系结构,并行与分布处理.

储规模如何变化,数据如何重新布局,对用户是透明的,用户总是可以不受影响地、可靠地访问数据.在 PB(10^{15}字节)级存储系统中,存储节点失效每天发生的概率非常大[1,2],为了保证存储数据不被丢失,必须提供相应的容错机制.因此,需要一个灵活、可伸缩、容错的数据布局算法来实现整个存储系统的数据管理,要求算法能够根据存储节点的权重均衡地分配数据对象,自动地适应存储节点的增加或减少,并且在存储节点发生变化时,需要重新布局的数据对象尽可能地少(即尽可能少地迁移数据).

目前的数据对象布局算法只能部分地满足上面的算法要求,如简单散列算法根据下式确定存储节点 i:$i=id \bmod n$,其中 id 为对象 ID,n 为系统的当前存储节点数.该方法在存储节点数不变时简单、有效,但是,当增加新的存储节点时,$n/(n+1)$的数据对象需要重新布局,而最优的需要重新布局的数据对象是其中的 $1/(n+1)$.Litwin 等人[3]提出的线性散列算法 LH*通过桶(bucket)的分裂(split)来实现数据规模的可伸缩自动增长,用散列函数序列实现数据对象的布局与重定位,缺点是没有考虑桶的权重,数据对象迁移只发生在分裂桶与目标桶之间,容易产生“访问热区”,重新布局后数据对象的分布是不均衡的.Choy 等人[4]提出一个均衡的数据布局算法,增加存储节点时能够最优地移动最少的数据对象,但该算法不支持按存储节点权重分配、数据对象副本以及存储节点的减少.Brinkmann 等人[5,6]提出基于单位区间分割使用伪随机分配数据对象的算法,通过重新分割区间支持存储规模增长及负载平衡,但迁移的数据对象不是最优的,且不支持数据对象副本;Honicky[7]提出一种均衡的数据对象布局快速算法,支持按存储节点的权重分配以及数据对象副本,但只支持节点的增加,不支持节点的减少,并且多次迭代计算随机数时间开销较大.而在实际应用中,一方面需要不断扩充新的存储节点,同时,必须淘汰快失效、维护成本过高的旧存储节点.

本文提出的算法使用确定性一致伪随机散列函数,将数据对象一致地、均衡地映射到单位区间,根据系统中存储节点的规模和权重将单位区间分割成不同长度的区间,并在区间与存储节点之间建立映射关系,通过两次映射确定数据对象的存储位置.当增加或减少存储节点时,不是重新分割整个单位区间,而是拆分当前的小区间成多个更小的区间,更新区间到存储节点之间的映射关系,使得需要迁移的数据对象最少.与现有的算法相比,本文提出的算法具有明显的优点:(1) 它能够按照存储节点的权重均衡地布局数据对象;(2) 它既能够适应存储节点的增加,也能够适应存储节点的减少,并且使得需要迁移的数据对象最少;(3) 经过简单扩充,它能够支持数据对象任意多个副本;(4) 客户端只需要维护一个数据量很小的全局区间映射信息和版本号(用于比较是否是最新的区间映射信息),并且该信息能够通过服务器端的反馈信息而自动更新,存储系统不需要维护一个庞大的数据对象映射表,所有数据对象的定位是经过客户端计算完成的,计算分布到客户端能够实现分散计算,提高了计算效率.

本文第 1 节给出描述系统和算法相关的术语.第 2 节提出动态区间链表的构造算法以及基于动态区间链表的支持权重的数据对象布局与重布局算法,并通过理论分析证明算法是均衡的、自适应的和迁移最优的,测试实验表明定位对象的计算开销小.另外,还提出支持多个对象副本的数据对象布局算法及相关讨论.最后是本文的结论.

1 系统和算法的定义

定义 **1**. 对象 ID 是标示数据对象的唯一标示符.

本文着重于数据对象布局算法.有关基于对象存储的存储系统见文献[8,9].

定义 **2**. H 为一个确定性一致伪随机散列函数[10],它将任意的数据对象 ID 一致地、均衡地映射到单位区间[0,1]中的某个实数.

定义 **3**. 目标存储系统由多个存储节点构成,存储节点依次编号为 0,1,2,…,通过增加或减少存储节点实现目标存储系统的伸缩.

定义 **4**. 存储节点 i 的权重(处理器能力、内存大小、存储容量等)为 w_i.

定义 **5**. 存储节点 i 包含的数据对象数目为 s_i.

定义 **6**. 若系统中任何两个存储节点所包含的数据对象数目与它们在系统中的权重成比例(即 $\frac{s_i}{s_j}=\frac{w_i}{w_j}$),

则系统称为负载均衡系统.

定义 7. 若根据某算法确定数据对象布局的系统是负载均衡系统,则称该算法是均衡的.

定义 8. 负载均衡系统在存储节点发生变化以后,若根据某算法重新布局后使得新的系统仍然是负载均衡系统,则称该算法是自适应的.

定义 9. 负载均衡系统在存储节点发生变化以后,若根据某算法重新布局达到新的负载均衡系统所需要迁移的数据对象最少,则称该算法是迁移最优的.

2 基于动态区间映射的数据对象布局算法

基于动态区间映射的数据对象布局算法的基本思想是,根据存储节点的权重将单位区间 [0,1) 分割成不同长度的区间,区间与存储节点建立映射关系,允许多个区间映射到同一个存储节点.映射到同一个存储节点的区间记录到一个区间链中,即在区间链与存储节点之间建立一个一一映射的关系,全体区间链构成一个区间链表.任何数据对象首先由 H 确定其映射的区间,由该区间确定其所在的区间链,从而确定对应的存储节点;当存储节点发生变化时,不需要重新分割整个单位区间,而是拆分当前的小区间成多个更小的区间或者重新更改小区间的映射节点,从而更新区间到存储节点之间的映射关系,其实质是引发局部区间的分割以及在不同区间链之间迁移,变化的只是区间链中的区间集合,区间链与存储节点的一一映射关系始终保持不变(如图 1 所示).这种做法能够使需要迁移的区间长度和最短,从而实现需要迁移的数据对象最少.用于计算定位数据对象的全局信息除了区间链表以外,还需要一个区间链表级别表示当前的区间链表版本,因为存储系统规模的变化不需要显式地通知到客户端,只有当客户端在请求或定位数据对象时,存储节点根据区间链表级别检查客户端使用的是否最新的区间链表,若不是,则更新该客户端的区间链表信息,从而保证存储系统各方使用相同的区间链表定位数据对象.

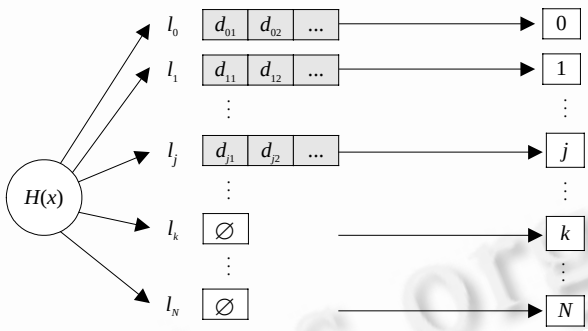

Fig.1 Dynamic interval mapping method

图 1 动态区间映射方法

2.1 初始区间链表 L_0

不失一般性,假定初始系统中的存储节点数目为 $k(k\geq 1)$,分别编号为 0,1,2,…,$k-1$.对任意的 $i(0\leq i\leq k-1)$,

$$a_i=\frac{\sum_{j=0}^{i} w_j}{\sum_{j=0}^{k-1} w_j},$$

令 $a_{-1}=0$,则得到实数序列:$0=a_{-1}<a_0<a_1<a_2<\ldots<a_{k-1}=1$,从而得到一系列的区间 $d_i=[a_{i-1},a_i)$ $(0\leq i\leq k-1)$.构造初始区间链表 $L_0=\{l_0,\ldots,l_{k-1},\ldots,l_N\}$,其中 $l_j=\begin{cases}\{d_j\}, & 0\leq j\leq k-1\\ \varnothing, & k\leq j\leq N\end{cases}$ ($\varnothing$ 表示空集,N 为存储系统可能的最大节点数目).

2.2 动态区间链表L_{i+1}

假定第 i 次存储节点变化后得到的区间链表 $L_i=\{l_0,\ldots,l_{k-1},\ldots,l_N\}$,其中,$l_j=\begin{cases}\{d_{j1},d_{j2},\ldots\}, & 0\le j\le k-1\\ \varnothing, & k\le j\le N\end{cases}$.根据 L_i 构造发生下一次存储节点变化后的 L_{i+1}.下面分两种情况加以讨论:

(1) 增加存储节点.由区间链表 L_i 定义可知,第 k 个以后的区间链包含的区间为空集,所以当前系统增加的下一个存储节点为第 k+1 存储节点,编号为 k,相应的容量权重为 w_k.对任意的区间链 l_j,遍历其包含的所有区间 $d_{j1},d_{j2},\ldots$,直到得到一个区间集合 D_j(如果需要,可以分割其中的区间,如图 2 所示),使得该区间集合的区间长度和为

$$\sum_{d_j\in D_j}|d_j|=\frac{w_j}{\sum_{i=0}^{k}w_i}.$$

保留这些区间在区间链 l_j 中,而将 l_j 中的其他区间迁移到区间链 l_k 中,从而得到区间链表 $L_{i+1}=\{l_0,\ldots,l_k,\ldots,l_N\}$,其中 $l_j=\begin{cases}\{d_{j1},d_{j2},\ldots\}, & 0\le j\le k\\ \varnothing, & k+1\le j\le N\end{cases}$.

(2) 去掉存储节点.假定当前系统去掉一个编号为 z 的存储节点($0\le z\le k-1$).对区间链 l_z,令其区间集合的区间长度和 $|D_z|=\sum_{d_j\in l_z}|d_j|$,遍历其包含的所有区间 $d_{z1},d_{z2},\ldots$,直到得到 $k-1$ 个区间集合 D_j(如果需要,可以分割其中的区间,如图 3 所示),使得对任意的 j,若 $0\le j<z$,则区间集合 D_j 的区间长度和为

$$\sum_{d_j\in D_j}|d_j|=\frac{w_j}{\sum_{i=0,j\ne z}^{k-1}w_i}\times|D_z|,$$

将 D_j 中的区间分别迁移到区间链 l_j 中;若 $z\le i\le k-2$,则区间集合 D_j 的区间长度和为

$$\sum_{d_j\in D_j}|d_j|=\frac{w_{j+1}}{\sum_{i=0,j\ne z}^{k-1}w_i}\times|D_z|,$$

将 D_j 中的区间分别迁移到区间链 l_{j+1} 中.去掉存储节点 z 后,保持存储节点编号的连续性,z 后面的存储节点编号将减 1,相应的区间链前移一行.从而得到区间链表 $L_{i+1}=\{l_0,\ldots,l_k,\ldots,l_N\}$,其中 $l_j=\begin{cases}\{d_{j1},d_{j2},\ldots\}, & 0\le j\le k-2\\ \varnothing, & k-1\le j\le N\end{cases}$.

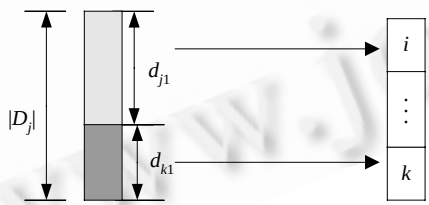

Fig.2 Adding storage node

图 2 增加存储节点

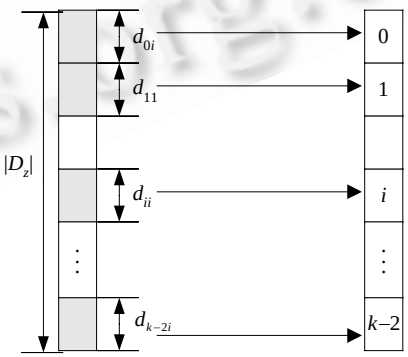

Fig.3 Removing storage node

图 3 去掉存储节点

2.3 数据对象映射算法

定义区间链映射函数 $F:L_i\to\{0,1,\ldots,N\}$,$F(l_j)=j(0\le j\le N)$.

对任意的数据对象 x,由 GetNode()函数确定其对应的存储节点:

int GetNode(数据对象 x,区间链表 L_i) {

if (在区间链表 L_i 找到区间链 l_j,使得 $H(x)\in d$,且 $d\in l_j$) then

return 存储节点编号 j

else

return −1

}

根据第 2.1 节的初始区间链表以及第 2.2 节的动态区间链表构造方法实现数据对象布局与重布局.

2.4 算法分析

2.4.1 数据对象分布

引理 **2.1**. 存储节点的数据对象数目与其对应区间链所包含的区间长度和成比例.

证明:因为 H 是一致伪随机散列函数,它将任意的数据对象一致地、均衡地映射到单位区间[0,1],所以映射到不同区间的数据对象数目与目标区间的长度成比例(统计意义上),而映射到同一个区间链中的区间的数据对象分配到同一个存储节点中,所以存储节点的数据对象数目与其对应区间链所包含的区间长度和成比例. □

定理 **2.2**. 任何存储节点所包含的数据对象数目与其对应的权重成比例.

证明:由引理 2.1,只需证明任何两个存储节点对应区间链所包含的区间长度和与它们的权重成比例即可.用数学归纳法证明.对任意的存储节点 p,q,设其对应区间链所包含的区间长度和分别为$|D_p|,|D_q|$.

(1) 初始时,i=0,

$$\frac{|D_p|}{|D_q|}=\frac{|d_p|}{|d_q|}=\frac{|a_p-a_{p-1}|}{|a_q-a_{q-1}|}=\frac{\dfrac{\sum_{j=0}^{p}w_j-\sum_{j=0}^{p-1}w_j}{\sum_{j=0}^{k-1}w_j-\sum_{j=0}^{k-1}w_j}}{\dfrac{\sum_{j=0}^{q}w_j-\sum_{j=0}^{q-1}w_j}{\sum_{j=0}^{k-1}w_j-\sum_{j=0}^{k-1}w_j}}=\frac{w_p}{w_q},$$

即存储节点 p,q 对应区间链包含的区间长度和与它们的权重成比例.

(2) 假定第 i 次存储节点变化后得到的区间链表 L_i 中,对任意的存储节点 p,q 对应区间链包含的区间长度和与它们的权重成比例.考虑发生下一次存储节点变化后得到的区间链表 L_{i+1} 中,分两种情况讨论:(a) 增加存储节点.根据动态区间链表的构造算法,若 $0\le p,q<k$,则

$$|D_p|=\frac{w_p}{\sum_{i=0}^{k}w_i},\quad |D_q|=\frac{w_q}{\sum_{i=0}^{k}w_i},\quad \frac{|D_p|}{|D_q|}=\frac{w_p}{w_q}.$$

若 p,q 中有一个为新增的第 k+1 个存储节点(编号为 k),不妨设 $q=k$.根据假设,在区间链表 L_i 中,任意的存储节点 j 对应的区间链 l_j 包含的区间长度和与其权重成比例,即

$$|D_j|=\frac{w_j}{\sum_{i=0}^{k-1}w_i}.$$

发生存储节点变化后,区间链 l_j 中迁移到区间链 l_k 中的区间长度和为 $\left(\dfrac{w_j}{\sum_{i=0}^{k-1}w_i}-\dfrac{w_j}{\sum_{i=0}^{k}w_i}\right)$.所以

$$|D_q|=|D_k|=\sum_{j=0}^{k-1}\left(\frac{w_j}{\sum_{i=0}^{k-1}w_i}-\frac{w_j}{\sum_{i=0}^{k}w_i}\right)=\frac{w_k}{\sum_{i=0}^{k}w_i},\quad \frac{|D_p|}{|D_q|}=\frac{|D_p|}{|D_k|}=\frac{w_p}{w_k},$$

即存储节点 p,q 对应区间链包含的区间长度和与它们的权重成比例.(b) 减少存储节点.根据动态区间链表的构造算法,在区间链表 L_{i+1} 中,区间链 l_p 包含的区间长度和包括原来的区间长度和及新迁移过来的区间长度和,$|D_p|=|D_{\text{old}}|+|D_{\text{new}}|$,根据假设,

$$|D_{\text{old}}|=\frac{w_p}{\sum_{i=0}^{k-1}w_i},\quad |D_{\text{new}}|=\frac{w_p}{\sum_{i=0,j\neq z}^{k-1}w_i}\times|D_z|.$$

对任意的存储节点 p,q,则

$$\frac{|D_p|}{|D_q|}==\frac{\dfrac{w_p}{\sum_{j=0}^{k-1}w_j}+\dfrac{w_p}{\sum_{j=0,j\neq z}^{k-1}w_j}\times|D_z|}{\dfrac{w_q}{\sum_{j=0}^{k-1}w_j}+\dfrac{w_q}{\sum_{j=0,j\neq z}^{k-1}w_j}\times|D_z|}=\frac{w_p}{w_q}.$$

即存储节点 p,q 对应区间链包含的区间长度和与它们的权重成比例.

综合(1)和(2)可知,存储节点 p,q 对应区间链包含的区间长度总是与它们的权重成比例,因此结论成立. □

由定理 2.2 的证明得知,基于动态区间映射的数据对象布局算法是均衡的,并且自动适应存储节点的增加和减少,数据对象重布局算法是自适应的.

定理 **2.3**. 存储节点变化后,需要迁移的数据对象数目是最少的.

证明:由引理 2.1,只需证明存储节点变化后,需要迁移的区间长度和是最小的即可,分两种情况讨论.

(1) 增加存储节点,假定增加第 k+1 个存储节点,编号为 k,其权重为 w_k.根据定理 2.2,任何存储节点所包含的数据对象数目与其对应的权重成比例,即迁移到区间链 l_k 中的区间长度和最少为 $\dfrac{w_k}{\sum_{j=0}^{k}w_j}$.而在动态区间链表的构造算法中,需要迁移的区间长度和为

$$\sum_{j=0}^{k-1}\left(\frac{w_j}{\sum_{i=0}^{k-1}w_i}-\frac{w_j}{\sum_{i=0}^{k}w_i}\right)=\frac{w_k}{\sum_{i=0}^{k}w_i},$$

即增加存储节点后,需要迁移的区间长度和是最小的.

(2) 减少存储节点,需要迁移的区间长度和就是该存储节点对应区间链所包含的区间长度和,而在动态区间链表的构造算法中恰恰只需要迁移该存储节点对应区间链所包含的区间,即减少存储节点后,需要迁移的区间长度和也是最小的.

综合(1)和(2)可知,存储节点发生变化后,需要迁移的数据对象数目是最少的. □

由定理 2.3 的证明可知,基于区间映射的数据对象重布局算法是迁移最优的.

2.4.2 计算复杂性

对任意的数据对象 x,确定它分布的存储节点需要遍历区间链表查找 $H(x)$所属的区间链,因此算法的计算复杂性依赖于区间链表中的区间总数,那么,区间链中的区间总数到底有多少?会不会随着存储系统规模的变化而增长得很快呢?从前面的动态区间链表构造算法中可以看到,无论是增加存储节点,还是去掉存储节点,都可能因为区间的分割而增加区间数量,但增加的区间数量最多为当前区间链的数量,即存储节点的数量.实际应用中不可能非常频繁地增加存储节点后又去掉存储节点(其中因为存储节点的失效而恢复存储节点数据不考虑为去掉存储节点和增加存储节点,不影响区间的增减),人们更多地考虑区间总数是否会随着存储系统规模的扩展而增长得很快?若忽略很少发生的因去掉存储节点而增加的区间数量,只考虑存储节点的增加,那么定理 2.4 给出了区间链表中区间总数的上界,它只与系统的初始存储节点数量以及增加存储节点的次数有关,而与数据

对象数量无关.相比系统中的数据对象数量,系统的初始存储节点数量以及增加存储节点的次数是非常小的.

定理 **2.4**. 若 n_0 为系统的初始存储节点数,t 是增加存储节点的次数,s 是区间链表中的区间总数,则有 $s \le \frac{1}{2}t \times (t-1) + (t+1) \times n_0$.

证明:用数学归纳法证明.

(1) 初始时,t=0,s=n_0,显然结论成立.

(2) 假定在第 t 次增加存储节点时,结论成立,即 $s \le \frac{1}{2}t \times (t-1) + (t+1) \times n_0$.那么,在第 t+1 次增加存储节点时,动态区间链表包含 n_0+t 个区间链,根据动态区间链表的构造方法,最多增加 n_0+t 个区间,则 $s \le \frac{1}{2}t \times (t-1) + (t+1) \times n_0 + n_0 + t = \frac{1}{2}(t+1) \times (t+1-1) + (t+1+1) \times n_0$,即结论仍然成立.

综合(1)(2),由数学归纳法可知结论总成立. □

在具体的实现中,可考虑用表或树数据结构,易知算法的计算复杂度为 $O(\log s)$.

为了对算法的真实性能进行定量测试,我们在多种配置下测试了 1 000 000 个数据对象的查找时间,从而计算得到单个数据对象查找的平均时间.存储系统的初始配置为 10 个存储节点,依次增加存储节点.图 4 显示出单个数据对象查找时间随存储节点数量的增加而缓慢增长.并且当存储系统规模扩展到超过 100 个存储节点时,单个数据对象查找时间也不超过 7μs,可见定位数据对象的速度是非常快的(测试机器配置为 Intel Pentium Ⅳ 1.4G).同时,图 4 也显示了数据对象查找时间随存储节点数量的增加而增长的速度明显低于线性函数与 *N*log*N* 函数.

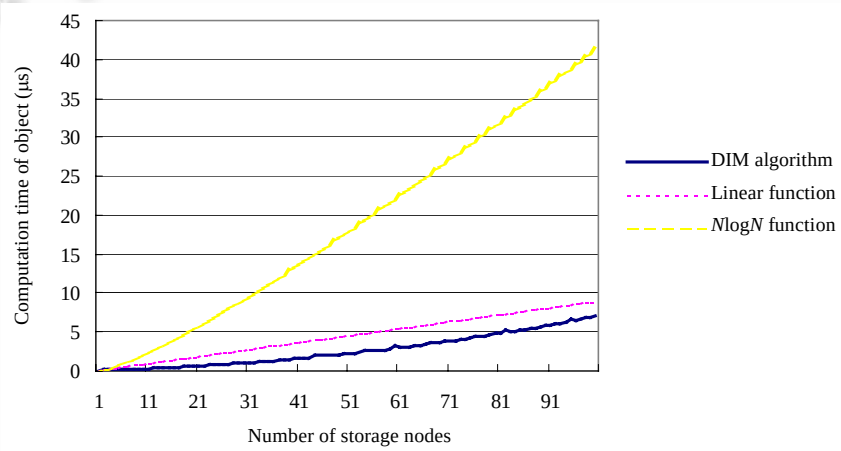

Fig.4 Time per lookup with number of nodes and compared to linear and *N*log*N* functions

图 4 数据对象查找时间增长曲线以及与线性函数、*N*log*N* 函数的比较

2.5 支持对象副本

修改 GetNode()算法,得到支持多个对象副本的数据布局算法.假定当前存储系统为 k 个节点,依次编号为 0,1,2,…,k–1.对任意的数据对象 x,其副本数 r 由 GetReplicaOfNode()函数确定相应的存储节点:

```
int GetReplicaOfNode (数据对象 x,副本数 r,区间链表 L_i) {
if (r>k)
    return  –1  //副本数 r 不超过当前存储系统最大节点数目
if (在区间链表 L_i 找到区间链 l_j,使得 H(x)∈d,且 d∈l_j) then
    return  存储节点编号(j+r) mod k
else
    return  –1
```

}

在大规模的存储系统中,节点失效的概率比较大,采用多个对象副本是保证数据可靠的有效措施[1,2],GetReplicaOfNode()算法始终依据动态区间映射基本算法,即根据区间链表 L_i 确定数据对象的存储节点,同时将其副本依次存放到其他存储节点,保证数据对象的各个副本总是分布到不同的存储节点中,并且数据对象的副本数目可以根据数据对象的重要性而有所不同,最多可以支持 k(当前存储系统最大节点数目)个副本.在支持对象副本的具体实现中,有 3 个问题需要注意:(1) 如果每个对象的副本数目不一样,则可能引起存储节点的数据对象数目与其权重不能完全成比例;(2) 在存储节点发生变化时,迁移数据对象时,若目标存储节点中已经存在该数据对象的副本,为保证数据对象副本数目不变,可考虑修正目标存储节点;(3) 在基于对象存储的大规模的存储系统中,对象的元数据信息保证在元数据服务器中,可以通过一个特殊的进程管理副本的迁移,而不影响系统的访问.这些问题需要我们进一步加以研究.

3 结 论

本文提出的基于动态区间映射的数据布局算法能够根据存储节点的权重均衡地分布数据对象,能够自动地适应存储节点的增加或减少,重布局算法使得需要迁移的数据对象最少,支持数据对象任意多个副本.理论分析表明,该算法具有统计意义上的均衡性、自适应性和迁移最优性.实验测试表明,定位对象的计算开销较小.

References:

[1] Ghemawat S, Gobioff H, Leung ST. The google file system. In: Larry P, ed. Proc. of the 19th ACM Symp. on Operating Systems Principles. New York: ACM Press, 2003. 19−22.

[2] Xin Q, Miller EL, Long DDE, Brandt SA, Schwarz T, Litwin W. Reliability mechanisms for very large storage systems. In: Moore R, eds. Proc. of the 20th IEEE/the 11th NASA Goddard Conf. on Mass Storage Systems and Technologies. Washington: IEEE Computer Society, 2003. 146−156.

[3] LitwinW, Neimat MA, Schneider DA. LH*—A scalable, distributed data structure. ACM Trans. on Database Systems, 1996,21(4): 480−525.

[4] Choy DM, Fagin R, Stockmeyer L. Efficiently extendible mappings for balanced data distribution. Algorithmica, 1996,16(2): 215−232.

[5] Brinkmann A, Salzwedel K, Scheideler C. Efficient, distributed data placement strategies for storage area networks. In: Miller G, ed. Proc. of the 12th ACM Symp. on Parallel Algorithms and Architectures (SPAA). New York: ACM Press, 2000. 119−128.

[6] Brinkmann A, Salzwedel K, Scheideler C. Compact, adaptive placement schemes for non-uniform capacities. In: Maggs B, ed. Proc. of the 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA). New York: ACM Press, 2002. 53−62.

[7] Honicky RJ, Miller EL. A fast algorithm for online placement and reorganization of replicated data. In: Dongarra J, ed. Proc. of the 17th Int'l Parallel & Distributed Processing Symp. Nice: IEEE Computer Society, 2003.

[8] Weber RO. Information technology—SCSI object-based storage device commands (OSD). Technical Council Proposal Document T10/1355-D, Technical Committee T10, 2004. http://www.t10.org/ftp/t10/drafts/osd/osd-r10.pdf

[9] Zhang YH, Zheng WM. A cluster storage architecture on network-attached object-based devices. Journal of Software, 2003,14(2): 293−299 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/293.htm

[10] Matsumoto M, Nishimura T. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. ACM Trans. on Modeling and Computer Simulation, 1998,8(1):3−30.

附中文参考文献:

[9] 张悠慧,郑纬民.基于网络附属对象设备的集群存储体系结构.软件学报,2003,14(2):293−299. http://www.jos.org.cn/1000-9825/14/293.htm