

基于一致性树分布的数据分布式存储方法

郭 栋^{1,2*}, 王 伟^{1,2}, 曾国荪^{1,2}

(1. 同济大学 计算机科学与技术系, 上海 200092; 2. 国家高性能计算机工程技术研究中心 同济大学分中心, 上海 200092)

(* 通信作者电子邮箱 gd@tongji.com)

摘 要:随着云计算和大数据技术的发展,传统的单一存储介质的数据存储方式已经不能满足大数据处理的需求,在这样的背景下,分布式数据存储得到了广泛的应用。然而,目前存在的几种分布式存储方式并不能够完美地满足分布系统的需求。为了更有效地实现数据的分布式存储和冗余备份,采用一种新的基于一致性树分布(CTD)的分布式存储方法,并提出基于该方法的备份策略,实现数据索引与存储位置的映射。该方案具有负载平衡、无单点故障问题、扩展性高且易于实现的优点。同时提出了基于一致性二叉树分布(CBTD)的应用方案。通过对应用系统实例的分析,验证该方法能够很好地满足分布式系统的数据平衡、负载均衡和扩展性需求。

关键词:分布式系统;分布式存储;一致性树分布;一致性二叉树分布

中图分类号: TP311.12 **文献标志码:** A

Distributed data storage method based on consistent tree distribution

GUO Dong^{1,2*}, WANG Wei^{1,2}, ZENG Guosun^{1,2}

(1. Department of Computer Science and Technology, Tongji University, Shanghai 200092, China;

2. Tongji Branch, National Engineering and Technology Center of High Performance, Shanghai 200092, China)

Abstract: With the development of cloud computing and big data, traditional single storage medium based data storage cannot meet the demands of large data processing, thus distributed data storage has been widely used recently. However, some existing distributed ways of storage cannot perfectly meet the needs of the distribution system. In order to achieve the distributed data storage and redundancy more effectively, a new distributed method and backup strategy based on Consistent Tree Distribution (CTD) was proposed to achieve the location mapping of data indexing and storage. The new method featured load balancing, no single point of failure, high scalability and easy programming. In addition, a method based on Consistent Binary Tree Distribution (CBTD) was also put forward. Analysis of the application system verifies that the method can satisfy the data balance, load balance and high scalability requirements of the distributed system.

Key words: distributed system; distributed storage; Consistent Tree Distribution (CTD); Consistent Binary Tree Distribution (CBTD)

0 引言

分布式存储系统是建立在网络之上的软件存储系统,将数据分散存储在多台独立的设备上,以提高系统的可靠性、可用性、存取效率和扩展性。对于一个分布式系统,核心在于其分布式算法,它直接影响系统的可用性、健壮性和扩展性,不同的分布式存储系统有不同的存储策略和方法。目前,已经存在很多分布式系统方案及其应用系统,如 Dynamo^[1]、HDFS^[2]等。主流的分布式存储算法有数据分割、哈希分布和一致性哈希分布^[3-4]算法等,它们各有优缺点,应用领域各异,如 Dynamo 就采用一致性哈希分布,这种分布式方法已经被广泛运用到实际的生产环境中。

然而,现有的几种分布式存储方式并不能够完美地满足分布系统的需求,如数据分割无法解决数据平衡问题,哈希分布无法解决系统扩展问题。本文提出一种称作一致性树分布(Consistent Tree Distribution, CTD)的分布式存储策略,并提出

基于该策略的数据分布式存储方法,旨在提供一种更有效、可靠且可扩展的分布式数据存储方案。与目前存在的分布式系统和算法相比,它既有数据分割算法的简单性,又有哈希算法的平衡性,同时具有一致性哈希算法的易扩展的优势,此外基于该方法的冗余备份策略“异步顺序备份”可进一步增强其可用性和安全性。

本文首先分析分布式存储系统的要求,然后结合目前存在的分布式存储方案提出需要进一步改进和解决的问题,进而引出基于一致性树分布的分布式存储方案;然后对一致性树分布的设计进行详细描述,并提出基于一致性二叉树分布(Consistent Binary Tree Distribution, CBTD)的应用系统方案;最后以实际应用系统分析该方法的有效性。

1 相关研究

对于有效的分布式系统设计,主要需要解决以下四个方面的问题:分布式算法、冗余备份及其一致性、系统扩展和单

收稿日期:2013-07-29;修回日期:2013-08-02。

基金项目:国家自然科学基金资助项目(61272107, 61202173, 61103068);上海市优秀学科带头人计划项目(10XD1404400);教育部网络时代的科技论文快速共享专项研究课题资助项目(20110740001);教育部博士点基金资助项目(20110072120017);浙江大学 CAD&CG 国家重点实验室开放课题资助项目(A1311);南京大学计算机软件新技术国家重点实验室开放课题资助项目(KFKT2012B24);同济大学中央高校基本科研业务费专项资金资助项目(0800219208);中国科学院模式识别国家重点实验室开放课题资助项目(201103187)。

作者简介:郭栋(1991-),男,上海人,主要研究方向:分布式系统、云计算;王伟(1976-),男,上海人,副教授,博士,CCF 会员,主要研究方向:并行计算、分布式系统;曾国荪(1964-),男,上海人,教授,博士生导师,CCF 会员,主要研究方向:计算机软件、分布式系统、信息安全。

点故障^[5]。目前存在的主流分布式算法有数据分割、哈希分布和一致性哈希分布,它们各具优劣,在不同的分布式场景得到广泛应用。

1.1 数据分割

数据分割分布一般建立在文件系统或者数据库系统之上,核心思想是将大量的数据人为地分割成多个部分存在不同的存储节点上。如数据库的分库和分表,本质上来说就是数据分割,通过将数据库分割成多个部分,分别存储在不同设备上以达到分布式存储的目的;如目前的很多社交网站或电子商务网站,它们的数据库非常庞大,数据表需要进行横向或纵向分割,然后存放在不同的网络节点上以提高效率和并发处理性能^[6-7]。数据分割方法的优势在于简单易行,然而它也存在一些问题:由于数据分割有明确的界限,而随着数据的增长,新数据都会出现在界限的一侧,导致数据难以平衡;另外,在实际情况下,热点数据很有可能会被分配在同一部分里,而冷数据往往被分在另外的服务器里,难以实现负载的均衡;此外,由于分割方式是人为确定的,当系统需要进行扩展时,需要重新修改分割规则以适应新的分布式环境,这使得系统难以扩展^[8]。

1.2 哈希分布

哈希分布则有效地解决了数据的均匀分布问题,通常使用的方法是根据数据索引的哈希值对节点数取模来确定数据位置,表示为 $\text{hash}(\text{key}) \% N$ 。哈希分布主要用于分布式缓存,如目前比较流行的 Memcache 以及类 Memcache 的缓存系统,其优势在于数据均衡存储,但是其容错性和扩展性不高,如果有节点失效或者要增加一个节点到存储集群,会导致映射算法中 N 的改变, N 的变化会使数据命中大规模失效,必须大规模迁移数据,其可用性和稳定性无法保证。

1.3 一致性哈希分布

一致性哈希分布^[4]选择具体的机器节点时,不只依赖于缓存数据的 key 的 hash 本身,而是机器节点本身也进行了 hash 运算。首先求出每个服务节点的 hash ,并将其配置到一个 $0 \sim 2^{32}$ 的圆环区间上;其次使用同样的方法求出数据 key 的 hash 值,也将其映射到这个圆环上;然后从数据映射到的位置开始顺时针查找,将数据保存到找到的第一个服务节点上。如果超过 2^{32} 仍然找不到服务节点,就会保存到第一个节点上。

一致性哈希最大限度地抑制了 hash 键的重新分布。另外为了取得更好的负载均衡的效果,往往在服务器数量比较少的时候需要增加虚拟节点来保证服务器能均匀地分布在圆环上。

虽然一致性哈希分布在一定程度上很好地解决了数据的均衡和扩展性问题,但是其存在两个缺陷:1)引入的虚拟节点并不能完全解决数据均衡问题,当对系统扩容时,必然又会破坏服务器的均匀分布,为了实现均匀分布,必须对称地引入很多虚拟节点,而引入这些虚拟节点必须重新调整相应的数据分布,会导致大量的数据拷贝工作;2)由于引入了虚拟节点,系统不能保证原数据和备份数据不在同一个物理节点上,如果原数据和备份数据恰好映射在了2个属于同一个物理节点的逻辑节点上,一旦该物理节点失效,会导致这2个逻辑节点同时失效,很可能导致数据丢失^[9-12]。

总而言之,目前存在的分布式算法仍然存在一些问题,对于上述问题,基于一致性树分布的分布式存储可以有效解决。

2 基于一致性树分布的数据存储方案

一致性树分布存储的核心在于将存储节点组织为树的逻辑

结构形式,其一致性在于树中各节点的一致性,借助树形结构的优点和特性,解决分布式存储系统的数据均衡、稳定和扩展性。

2.1 基本思想

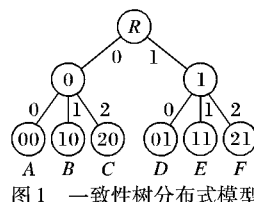
一致性树分布的基本思想在于分层分组,即将所有存储节点分布在不同层次的组里,数据按照不同层次定位到不同组里,从而完成映射过程。而分层分组可以用树形结构完美表示,从而形成一致性树分布式结构。

对于需要存取的数据,则依次在树的不同层次中定位所在的节点,即可确定其位置。

2.2 一致性树分布模型

一致性树分布模型用一棵多叉树表示,本质上来说是以分支关系定义的层次结构,对于某一特定的树形模型,在每一层次,节点都被分为多个互不相交的有限集 T_1, T_2, \dots, T_n , 其中每一个集合本身又是一棵树,从而将所有存储节点分到不同层次的不同组里。

在一致性树分布模型中,每个叶子节点对应实际的物理存储节点,每个内部节点为叶子节点的分组集合,如图1所示为一个包含6个节点的存储集群逻辑图。对于每个节点,都可以用唯一的一个数字序列定义,它是由该节点在树中的位置决定的,从右到左依次代表从根到该节点所经过的路径号(子树从左到右编号 $0, 1, 2, \dots$),如图1中B节点的编号 $\{1, 0\}$,表示从根节点到B需要经过路径0和路径1。那么,该数字序列也就唯一确定了该节点在树中的位置,当所有叶子节点的编号时,这棵树的逻辑结构也就确定了,二者之间是单射关系,所以,可以简单地把这样的树抽象成为一个二维数组,即各节点数字序列的列表,如代码1所示为图1中树的代码表示。



完整的数据映射过程可以表示为:

$$\text{Map}(\text{Hash}(\text{key}), \text{tree}) \quad (3)$$

这样,数据的定位就可以简单抽象为以树结构和数据 key 为参数的映射算法,一致性树分布的核心就在于该 Map 函数。基于上述定义和规范,下面对映射算法作详细的分析描述。

2.3 映射算法

一致性树分布的映射算法并不复杂,关键在于每一层的数据定位,从根遍历到叶子。对于某 id 数据到存储服务器的映射算法,即上述的 Map 算法,其过程如下:

1) 首先从根节点 root 开始,计算 $\text{id} \% \text{root.Degree}$,得到对应的子节点 node ,如果子节点是叶子节点,则返回该节点;否则,进行步骤2)。

2) 计算 id 在该子节点中的编号 inner_id , inner_id 理解为如果该 id 前的数据都存在,该 id 数据在该节点的位置。

3) 以 node 作为新的 root ,重复步骤1)。

对于 Map 函数,使用伪代码描述如代码2所示,其算法模型类似折半查找过程,算法时间复杂度为 $O(\log(n))$ 。

代码2 Map 函数。

```
node * Map(id, tree){
    node = tree.root;
    param = 1;
    inner_id = id;
    While (node not leaf){
        param * = node.Degree;
        node = node.Children[inner_id % node.Degree];
        inner_id = id/param;
    }
    return node;
}
```

该 Map 函数以 $\text{Hash}(\text{key})$ 和 $\text{struct} * \text{tree}$ 作为参数,返回对应的存储节点信息。上述步骤的关键在于计算 inner_id ,在描述 inner_id 的计算方式之前,需要定义节点 node 的度积 (Degree Product, DP): 对于某节点 node 的所有父辈节点 $\text{node}_1, \text{node}_2, \dots, \text{node}_n$:

$$\text{DP}(\text{node}) = \prod_{i=1}^n \text{node}_i.\text{Degree} \quad (4)$$

度积表示节点所有父辈节点的度的积,在一致性树分布式中其值为该节点中数据 id 的周期,例如图1中 $\text{DP}(A) = 3 \times 2 = 6$ 。

因此数据在某节点 node 的 inner_id 算法如下:

$$\text{inner_id} = \left\lfloor \frac{\text{id}}{\text{DP}(\text{node})} \right\rfloor \quad (5)$$

可见,一致性树分布的映射算法本身并不复杂,且很容易用程序实现。

2.4 一致性树分布冗余备份

2.4.1 备份策略

冗余备份的方法很多,关键在于确保备份和主数据都不在同一节点上。本文提出一种基于CTD的简单有效的备份策略,称之为“异步顺序备份”:

在一个具有 m 个存储节点的一致性树分布式系统中,对某 id 数据产生 n 个备份, n 必须满足如下条件:

$$n \leq \min(\text{DP}(\text{node}_1), \text{DP}(\text{node}_2), \dots, \text{DP}(\text{node}_n)) - 1 \quad (6)$$

确定最大备份数后,将该数据异步备份到 $\text{Map}(\text{id} + 1, \text{tree})$, $\text{Map}(\text{id} + 2, \text{tree})$, \dots , $\text{Map}(\text{id} + n, \text{tree})$ 所对应的节点上。

对于上述 n 的限制可以确保备份数据以及原数据都不在相同节点上,该结论很容易采用反证法证明,在 n 满足上述条件的前提下,加入有任意2个或2个以上的数据在同一节点 node ,那么该节点的数据 id 周期一定小于 $n + 1$,即

$$\text{DP}(\text{node}) < n + 1 \quad (7)$$

将上述(6)和(7)两个不等式相加可得:

$$\text{DP}(\text{node}) < \min(\text{DP}(\text{node}_1), \text{DP}(\text{node}_2), \dots, \text{DP}(\text{node}_n))$$

这显然与上述条件矛盾,故 n 个备份数据和原数据一定不在同一节点。

另外,采用异步顺序备份可以保证备份数据整体均匀分布,不会出现某节点存储了大量的备份数据而其他节点没有备份数据的现象。这一点也很容易递归证明:由于原数据 id 是均匀分布的,那么 $\text{id} + 1$ 的数据必然也均匀分布,同理 $\text{id} + 2, \text{id} + 3, \dots, \text{id} + n$ 的数据也是均匀分布的。

2.4.2 备份数据的一致性

有了备份策略后,需要关注一致性的问题,这也是分布式系统的最基本问题,常用的解决方案有:

1) 单点读写,数据IO都在同一个主节点进行,IO成功后立即返回,数据异步备份到备份节点,只有当主节点失效时,IO才会切换到备份节点。对于一致性树分布式,如果使用异步顺序备份策略,正常情况下IO请求都发往直接映射的主存储节点,当主节点失效时,IO请求发往 $\text{Map}(\text{id} + 1, \text{tree})$, $\text{Map}(\text{id} + 2, \text{tree})$, \dots , $\text{Map}(\text{id} + n, \text{tree})$ 对应的节点。

2) Dynamo中使用的NWR法则^[1],在一致性树分布式中仍然适用,关于NWR的具体描述见参考文献[1]。

2.5 一致性树分布的高扩展性和可缩容性

2.5.1 扩展性

一致性树分布的扩展采用节点分裂的方式进行,这也是一致性树分布的一个显著特点。以图1中的节点A为例,当A的容量将要不足或者IO压力较大时,可以将A节点分裂为多个节点以降低单个服务器压力,如图2所示。

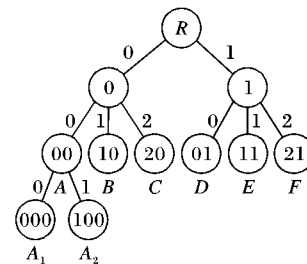


图2 一致性树分布式的扩展:节点分裂

节点分裂需要将被分裂节点的数据以及备份数据拷贝到新增加的节点上,数据拷贝方法如下:

假设将节点A分裂成 n 个节点 A_1, A_2, \dots, A_n , A节点当作 $A_i (1 \leq i \leq n)$ 节点使用,需要以下步骤:1) 遍历A中的所有数据,计算 $t = \lfloor \text{data.id} / \text{DP}(A) \rfloor \% n$; 2) 如果 $t \neq i$,将数据和备份拷贝到节点 A_i ; 3) 数据复制完成后,更新树形结构的二维数组,完成系统扩展任务。

该扩展方法影响的数据范围仅有被分裂节点的数据,其余节点无任何变化,为最小化影响,扩展后各节点仍然是等价的,这是一致性树一致性的重要体现。

2.5.2 缩容性

在某些情况下,如数据被删除或者被迁移到其他地方,存储能力远大于需求时,可以对一致性树分布式系统缩容,以释放空闲节点。

一致性树分布式的扩容和扩展方式完全相反,以图 1 的结构为例,如果需要将 A_2 释放,需要以下几个步骤:首先将 A_2 的数据以及备份复制到 A_1 ;然后将 A 替换成 A_1 ,即将 A_1 的数字序列 $\{0,0,0\}$ 改为 A 的数字序列 $\{0,0\}$;最后从树的二维数组中删除 A_2 的对应项。这样,节点 A_2 对应的物理节点就被释放了,影响的数据仅有 A_2 中的数据,依然是最小化影响。

2.6 一致性树分布的虚拟节点

虚拟节点在一致性树分布式中并不多用,由于一致性树分布可以确保在同一层节点上的数据均匀分布,在采用对称树结构时,不存在数据不均匀分布的情况。当然,在各节点存储能力有差别的情况下,也可以使用虚拟节点,也就是树形结构中多个叶子节点对应同一个物理节点,通过该方法可以提高系统利用率。但是,虚拟节点同样有类似一致性哈希分布的缺点,由于数据的可靠性是靠不同的存储介质备份,虚拟节点与该理念违背,仍然存在因失效而丢失数据的可能性,所以,在实际生产环境中虽然可以使用但是并不推荐。

2.7 一致性树分布的单点问题

在 2.2 节中将树抽象成一个二维数组,那么只需维护一个二维数组即可获得全局分布的信息。利用这一点,只需在每个节点都同步这样一个二维数组即可具有全局视图,任何一个节点都可以用来计算映射。在实际应用系统中,可以通过如心跳检测等方式保证全局视图的同步和更新。对于一致性树分布式方案,并不存在单点问题,所有节点都是等价的,这也是一致性树的一致性所在。

3 应用系统方案

3.1 一致性二叉树分布

前面对一般情况下一致性树分布式存储进行了介绍,其中,一致性二叉树分布(CBTD)是一致性树分布(CTD)的特例,也是其中最简单的形式,具有其独特性和优势。在使用二叉树结构作为逻辑存储结构时,映射和计算处理方式将变得相当简单,特别适合实际的应用系统。

在计算机内部,数据都是以二进制方式存储的,可以充分利用这个特点,将该特性巧妙地运用到一致性二叉树分布式存储上。这里以图 3 所示的二叉树存储结构为例作进一步说明。

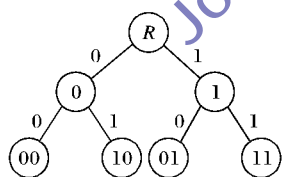


图3 一致性二叉树分布结构

在图 3 所示情况下,可以用一个整数 Seq 来代替之前定义的节点数字序列,该整数作为每台服务器的编号,如图 3 中叶子节点中的数字,从最低位到最高位依次对应从根节点到该节点所途径的路径序号。

当某数据 $data$ 要存取时,以 $hash(data.key) = 5$ 为例,按照 2.3 节的映射算法很容易得到其对应的服务器 01。然而,更为简单的方式是直接取 5 对应的二进制 1001 的后 2 位就是对应的服务器编号。同样道理, $hash(data.key) = 7$ 二进制为 111,则对应于 11 号服务器。计算机对二进制的计算和逻辑处理能力是最擅长的,这样的二叉树结构存储系统也会比多叉树系统的效率和性能要高。所以,对于最佳实践的应用系统,使用二叉树分布式存储结构能够降低系统复杂性,同时提高系统整体性能。

对于一致性二叉树,如果使用顺序查找,其算法时间复

杂度为 $O(n)$;当然,如果对一致性树的节点存储方式进行优化排序,可以使用更加高效的检索方式。如折半查找,则算法时间复杂度变为 $O(\log(n))$;对于大规模的节点数量较多的系统,如果是对称树模型,若使用哈希表做映射,可以实现 $O(\log(1))$ 的算法时间复杂度。总之,CBTD 算法过程更为简单灵活,从而可以进一步提高算法效率。

对于一致性二叉树分布结构的系统扩展性,与之前描述的方式一样,只是将母节点一半的数据和备份数据复制到新节点上。

3.2 使用有序整型编号

在 2.2 节中定义了将 key 映射为整型的 hash 函数,其作用是将无序的编号(如字符串)均匀分散。那么,如果数据编号本身是有序的,如数据库自动增长字段,那就没有必要再使用 hash 进行分散,也不必考虑 hash 的冲突问题,其本身的顺序已经能够保证在树形结构中的均匀分散,可以直接使用 key 作为一致性树分布中的数据 id 。所以,在具体的应用项目中,推荐使用顺序编号的整型 key 作为数据索引,这样能够进一步提高分布式系统的效率。

3.3 双份备份

在实际应用系统中,公认比较安全的备份数是 2 份备份,在使用一致性二叉树分布式存储时,更容易实现。

根据 2.4 节中介绍的异步顺序备份策略,必须满足式(6),取 $n = 2$ 可得:

$$\min(DP(node_1), DP(node_2), \dots, DP(node_n)) \geq 3 \quad (8)$$

而对二叉树分布式的任意叶子节点 $node$,其度积由其所在的层 l 唯一决定,即对任意叶子节点 $node$:

$$DP(node) = 2^{l-1} \quad (9)$$

由式(8)和(9)可得 $l \geq 3$,所以只需要保证所有叶子的层数至少为 3 即可保证双份备份的实施,而对应图 3 中所示的结构,则是实施备份的最小系统,在实际应用系统中这样的系统是很容易实现的。

4 实验结果与分析

4.1 实验系统与测试环境

基于上述方案,开发了一个基础的分布式块设备(Distributed block device)的应用实例。该应用提供类似 Amazon EBS 的功能,由一个一致性二叉树分布式系统提供基础存储,在客户端以块设备的形式挂载。该块设备在客户端的使用和普通块设备(如硬盘)完全相同,用户可以自己分区,自己装文件系统。而在具体存储方面,其数据都存在一致性树分布式系统中,并有异步的冗余备份。

分布式块设备被分为 1 MiB 大小的 unit,对各个 unit 从 0 开始编号,作为数据唯一索引 id 。

在一致性树分布式系统中,采用 4 台物理节点组成 3 层二叉树分布结构,即图 3 所示结构,各节点信息如表 1 所示。

客户端根据定义的树形结构二维数组在内存中建立树,在进行数据存取操作时,计算相应的 $unit_id$ 对应的存储节点,然后连接对应的节点进行 IO 操作。

表 1 分布式块设备节点信息

Name	IP	Port	Seq
Node1	192.168.1.100	8900	00
Node2	192.168.1.101	8900	10
Node3	192.168.1.102	8900	01
Node4	192.168.1.103	8900	11

4.2 测试方案

本文建立了一个 64 MiB 大小的块设备,使用代码 3 中的

命令安装 ext4 文件系统,并挂载到/mnt 目录下,写入一个 50 MiB 的空文件,再将其复制到本机。

代码 3 测试命令。

```
mkfs. ext4 /dev/dbd
```

```
dd if = /dev/zero of = /mnt/test bs = 1024 count = 50
```

```
cp /mnt/test ./
```

该测试包含数据读取和写入,通过查看各节点的数据量和 IO 次数来验证二叉树分布的有效性。

然后,对 Node4 进行扩展,将其分裂为 2 个节点 Node4-0 和 Node4-1,其中 Node4-0 就是扩展前的 Node4, Node4-1 为新增加的节点,通过此方法以验证其扩展性。

4.3 测试结果分析

基于上述测试方案,各节点的数据和备份的分布情况如图 4 和图 5 所示,节点扩展后的数据分布情况如图 6 所示。

图 4 表明,各节点数据分布均匀,各节点数据量为 16 MiB,备份数据为 32 MiB,并无数据倾斜现象,备份数据也均匀分布,证明了一致性树分布的数据均衡特性。图 5 显示了各节点的文件读写次数,实际上与客户端向该节点发送的 IO 请求次数成正比,从结果中可以看到各节点 IO 次数基本相当,可以很好地实现负载均衡。图 6 显示了对 Node4 进行扩展后各节点的数据分布情况,可以看到,Node4 的一半数据和备份数据被分配到了 Node4-1 上,其他节点并无任何变化和影响,受影响数据为最小范围。

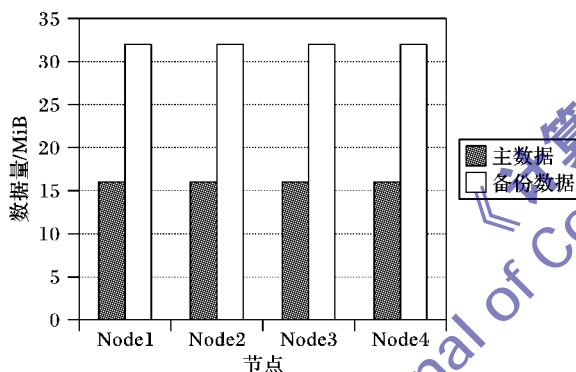


图4 节点扩展后的数据分布

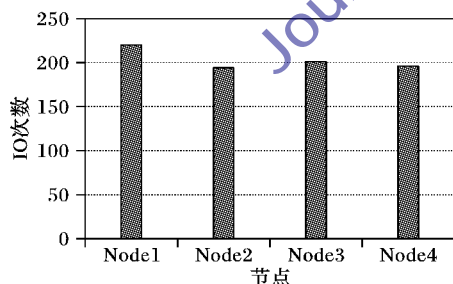


图5 各节点 IO 次数

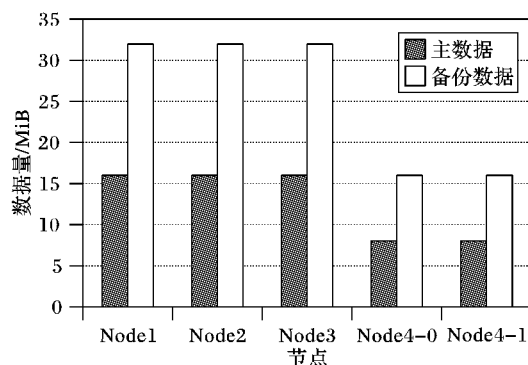


图6 对 Node4 进行扩展后各节点的数据分布

上述实验以典型的分布式块设备系统验证了一致性树分布的数据均衡和扩展性,当然,在实际的大规模应用系统中,其优势将更加突出,效果也将更加明显。

5 结语

本文提出了一种新的基于树形结构的分布式存储方案,相比传统的分布式方案,该方案能够提高可靠性、可用性和扩展性,同时具有实现简单的优点。本文分析了一致性树分布式(CTD)的通用模型和算法,同时对于实际的应用系统,给出了应用系统的实施方案一致性二叉树分布(CBTD),该方案可用于分布式文件系统、分布式块设备或分布式数据库等应用系统,具有广阔的应用前景。但是,针对一致性树分布还有很多问题在未来需要解决,如集群系统的通信和同步问题、系统扩展时数据一致性问题等。

参考文献:

- [1] DECANDIA G, HASTORUN D, JANPANI M, *et al.* Dynamo: Amazon's highly available key-value store [C]// SOSP '07: Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles. New York: ACM, 2007: 205-220.
- [2] Hadoop: Open source implementation of MapReduce [EB/OL]. [2013-05-01]. <http://lucene.apache.org/hadoop>.
- [3] TANENBAUM A S, STEEN M V. Distributed systems: principles and paradigms [M]. 2nd ed. Upper Saddle River, New Jersey: Pearson Prentice-Hall, 2006.
- [4] KARGER D, LEHMAN E, LEIGHTON T, *et al.* Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web [C]// STOC '97: Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing. New York: ACM, 1997: 654-663.
- [5] 王伟, 曾国荪. 构造基于信任机制的自组织资源拓扑[J]. 计算机研究与发展, 2007, 44(11): 1849-1856.
- [6] SINGHAL M. Deadlock detection in distributed systems [J]. Computer, 1989, 22(11): 37-48.
- [7] GLIGOR V D, SHATTUCK S H. On deadlock detection in distributed systems [J]. IEEE Transactions on Software Engineering, 1980, 6(5): 435-440.
- [8] MENACSE D A, MUNTZ R R. Locking and deadlock detection in distributed data bases [J]. IEEE Transactions on Software Engineering, 1979, 5(3): 195-202.
- [9] YANG S, de VECIANA G. Size-based adaptive bandwidth allocation: Optimizing the average QoS for elastic flows [C]// INFOCOM 2002: Proceedings of the Twenty-first Annual Joint Conference of the IEEE Computer and Communications Societies. Washington, DC: IEEE Computer and Communications Societies, 2002, 2: 657-666.
- [10] GARCIA-LUNA-ACEVES J J, BEHRENS J. Distributed, scalable routing based on vectors of link states [J]. IEEE Journal on Selected Areas in Communications, 1995, 13(8): 1383-1395.
- [11] LU J, TUMER J. Efficient mapping of virtual networks onto a shared substrate, WUCSE-2006-35 [R]. St. Louis: Washington University in St. Louis, Department of Computer Science And Engineering, 2006.
- [12] CHIANG M, ZHANG S, HANDE P. Distributed rate allocation for inelastic flows: Optimization frameworks, optimality conditions, and optimal algorithms [C]// INFOCOM 2005: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Washington, DC: IEEE Computer and Communications Societies, 2005, 4: 2679-2690.