Journal of Computer Applications
计算机应用,2012,32(3):620-624
ISSN 1001-9081
CODEN JYIIDU
2012-03-01
http://www.joca.cn

文章编号:1001-9081(2012)03-0620-05
doi:10.3724/SP.J.1087.2012.00620

大规模云存储系统副本布局研究

董继光*,陈卫卫,田浪军,吴海佳
(解放军理工大学 指挥自动化学院,南京 210007)
(*通信作者电子邮箱 pladjg@163.com)

摘　要:在基于副本冗余机制的大规模云存储系统中,以往的副本布局算法只能部分地满足副本布局中高可靠、高可扩展并且高效的要求,针对这一问题,提出了一种基于分组和一致性哈希的副本布局算法。首先,将关联性高的存储节点进行分组;然后,通过改进的一致性哈希算法将同一数据对象的多个副本分别分配到不同分组中;最后,再通过改进的一致性哈希算法将分配到各分组的数据副本放置在组内对应的存储节点上。理论分析可知,该方法大大提高数据的可靠性。仿真结果表明,该算法能满足副本布局的均衡性、自适应性要求,并能在几十微秒内完成副本定位。

关键词:云存储;副本布局;一致性哈希算法;分组;可靠性
中图分类号:TP301.6;TP302　　**文献标志码**:A

Replica placement study in large-scale cloud storage system

DONG Ji-guang*, CHEN Wei-wei, TIAN Lang-jun, WU Hai-jia
(*Institute of Command Automation, PLA University of Science and Technology, Nanjing Jiangsu* 210007, *China*)

Abstract: In the large-scale cloud storage system based on copy redundancy, previous layout algorithm can only partially meet the requirements of high reliability, high scalability and high efficiency in the replica layout. To solve this problem, this paper proposed a Replica Placement algorithm based on Grouping and Consistent Hashing (RPGCH). The storage nodes were classified into different groups by their correlativity, then the replicas of one object were assigned in different groups by consistent hashing algorithm, after that each replica was placed into corresponding storage node in the group by consistent Hashing algorithm. The theoretical analysis proves that the reliability of data is improved. The simulation results show that RPGCH can assign data evenly among storage nodes and adapt well with the changing scale of cloud storage system. Moreover, RPGCH is time efficient with little memory overhead.

Key words: cloud storage; replica layout; consistent hashing algorithm; grouping; reliability

0　引言

Jim Gray[1]曾提出一个经验性定律:在网络环境下,每18个月产生的数据量等于有史以来数据量的总和。据IDC调查统计,从2006年到2010年,全球产生的数据总量增长了6倍以上,从161 EB增加到988 EB(1 EB = 10^{18} B),而且数据的增长速度还在持续增加[2]。由于基于大规模PC集群构建的高可伸缩云存储系统具有海量并行的数据存储和处理能力,因此它成为PB级海量数据存储问题的有效解决方案。在网络运营商的数据中心,数据规模可达PB级甚至EB级,因此数据对象的数量可达百亿级别,甚至更高。如何将数据对象高效地分布到云存储系统中,并能满足云存储系统的节点数量大、系统伸缩性强、数据可靠性要求高、节点存储能力差异大等方面的要求成为一个非常具有挑战性的课题。

一种被广泛采用的数据布局方式是通过集中式的存储目录来定位数据对象的存储位置。这种方法可以利用存储目录中存放的存储节点信息,将数据对象的多个副本放置在不同机架上,这样可大大提高系统的数据可靠性。谷歌文件系统(Google File System, GFS)[3]、Hadoop分布式文件系统(Hadoop Distributed File System, HDFS)[4]等著名的分布式文件系统都采用了这种数据布局方式,然而,基于集中式存储目录的数据放置方式存在以下两个缺陷:1)随着存储目录的增长,查找数据对象所需的开销也会越来越大;2)为提高数据对象的定位速度,一般情况下都会将存储目录存放在服务器内存中,对于PB级的云存储系统来说,文件的数量可能达到上亿级,这导致存储目录将会占用上百吉字节的内存。因此,当数据对象数量达到上亿级别时,基于集中式存储目录的数据放置方法在存储开销和数据定位的时间开销上都是难以接受的,此外,还会大大限制系统的扩展性。Weil等[5]研究指出,基于集中式存储目录来决定数据对象布局的做法是大规模存储系统实现高可扩展的主要瓶颈。

另一种副本布局方法是基于哈希算法的副本布局方法,它完全摒弃了记录数据对象映射信息的做法。基于哈希算法的副本布局方法需要满足以下要求:1)均衡性。根据节点权重为存储节点分配数据对象。2)动态自适应性。当系统中的节点数量发生变化时,需迁移的数据量应该尽量少。3)低性能开销。4)高效性。确定副本位置所需的时间开销尽可能小,理想情况下为$O(1)$。Karger等[6]提出了一致性哈希算法,该算法能适应节点数量的动态变化,但它只适用于存储节点同构的情况,当节点的存储容量和处理能力有差异时,数据将不能够均匀地分布到系统当中。Brinkmann等[7]提出的副本布局算法递归地将一个数据对象的k个副本放置在不同存

收稿日期:2011-09-19;**修回日期**:2011-11-16。
基金项目:国家自然科学基金资助项目(60603029);国家863计划项目(2008AA01A309)。
作者简介:董继光(1986-),男,河南周口人,硕士研究生,CCF会员,主要研究方向:分布式网络存储、云存储;　陈卫卫(1967-),女,四川隆昌人,教授,CCF会员,主要研究方向:软件工程、云计算、分布式网络存储;　田浪军(1985-),男,陕西渭南人,硕士研究生,CCF会员,主要研究方向:分布式数据库;　吴海佳(1986-),男,江苏南通人,博士研究生,CCF会员,主要研究方向:分布式网络存储、分布式文件系统。

储节点上,可是在系统节点数量发生变化时,该算法需要迁移的数据量是最优情况下的 k^2 倍。Honicky 等[8]提出了一种叫作 RUSH 的副本布局算法,后来,Weil 等[9]提出了一种叫作 CRUSH 的副本布局算法,它是 RUSH 算法的改进,然而它并不支持单节点的加入和删除。Xiao 等[10]提出了一种 RSEDP 副本布局算法,该算法将性能接近的存储节点划分为一类,在类内使用一致性哈希算法,但该算法不支持频繁单个存储节点的加入和删除。Amazon[11]的云存储系统使用了 Dynamo 架构,这种架构利用改进的一致性哈希算法的优点,为物理存储节点创建适量的虚拟节点,保证了系统的高可扩展性,数据布局的均衡性、动态自适应性,并能快速定位数据对象位置,然而它却不能将数据副本分配到不同机架上,当出现机架上的电源或交换机失效时,将会引起所有副本都在同一机架上的数据对象丢失,因此在相同副本数量的条件下,Dynamo 架构比 GFS 的数据可靠性低。

综合分析 GFS 和 Dynamo 的优点,本文提出了一种基于分组和一致性哈希的副本布局(Replica Placement based on Grouping and Consistent Hashing, RPGCH)算法。该算法根据存储节点的相关性进行分组,在每个分组内只保存同一数据对象的一个副本,这就大大提高了系统数据可靠性;在组间和组内分别使用一致性哈希算法进行副本布局,既保证了系统的高可扩展性,又能满足副本布局算法在均衡性、动态自适应性以及性能开销低等方面的要求。

1　RPGCH 算法

RPGCH 算法的设计目标是在保证均衡性、动态自适应以及低性能开销的前提下,让同一数据对象的多个副本放置到不同的灾难域中,这样可大大提高数据的可靠性。灾难域的划分可根据实际情况确定,划分的原则是同一个灾难域中的存储节点具有较高的关联性,可认为一个交换机或电源为一个灾难域,也可认为一个机房是一个灾难域,甚至可认为一个数据中心为一个灾难域,灾难域的划分与数据的可靠性要求有关,将每个灾难域看作一个存储节点分组。RPGCH 算法的具体步骤如下:

1)将云存储系统中的存储节点划分为不同的节点分组;

2)对于要存放到云存储系统中的数据对象,通过改进的一致性哈希算法,将数据对象的不同副本放置到不同的节点分组中;

3)对于分配到某一节点分组的数据副本,通过改进的一致性哈希算法,将该副本存放节点分组的某一节点上。

1.1　算法相关定义

如果将存储节点和数据副本分别看作一个集合,则云存储系统的副本布局问题可抽象为寻找一种从数据副本到存储节点的映射问题,并且解决该问题的映射算法满足基于哈希算法的副本布局要求。为便于问题描述,首先做出以下符号定义。

数据副本集合 R:$R = \{r_{(i,j)} \mid 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant k\}$,其中 m 表示不同数据对象的个数,k 表示单个数据对象的副本个数,如 $r_{(i,j)}$ 表示第 i 个数据对象的第 j 个副本。

组内节点集合 G_i:$G_i = \{d_{(i,j)} \mid 1 \leqslant j \leqslant C_i\}$,$d_{(i,j)}$ 表示第 i 个分组的第 j 个节点,$\| G_i \| = C_i$,C_i 为第 i 组中的节点个数。

分组集合 G:$G = \{G_i \mid 1 \leqslant i \leqslant C\}$,$\| G \| = C$,$C$ 为分组个数。

分组权重集合 GW:第 i 个节点分组与全体分组的相对权重为 gw_i,则 $GW = \{gw_1, gw_2, \cdots, gw_C\}$,且 $\sum_{i=1}^{C} gw_i = 1$,$\| GW \| = C$。

组内节点权重集合 NW_i:第 j 个节点与该节点所在节点分组的相对权重表示为 $nw_{(i,j)}$,则 $NW = \{nw_{(i,1)}, nw_{(i,2)}, \cdots, nw_{(i,C_i)}\}$,且 $\sum_{j=1}^{C_i} nw_{(i,j)} = 1$,$\| NW_i \| = C_i$。

哈希空间 O:$O = \{r \mid 1 \leqslant r \leqslant M, r \in \mathbf{N}\}$,它是数据对象以及存储节点的映射空间,样本空间中的元素按从低到高的顺序看作一个虚拟的首尾相接的环。

1.2　组间副本布局机制

根据一致性哈希算法的思想,组间副本布局算法利用哈希函数 h 将存储节点分组集合 G 映射到哈希空间 O 中。由于 G 中元素的权重有差异,为解决这个问题,本文引入了虚拟节点的概念,每个虚拟节点都属于某一个实际的物理节点分组,每个物理节点分组根据其权重差异拥有不同数量的虚拟节点,所有的虚拟节点性能相同,并通过哈希算法将虚拟节点随机地分布到哈希空间。组间副本布局通过下面 3 个函数实现。

1) 分组映射函数 GroupOnRing(G,GW,v_1)。该函数的功能是通过哈希运算,得到 G 中每个元素的虚拟节点在哈希空间中的映射值。输入参数 G 为分组集合,v_1 为权重最小分组所具有的虚拟节点个数,输出用 HG 表示,它是 G 中所有元素的虚拟节点在哈希空间上的映射值的集合。对于 G 中的任意元素 G_i,其映射结果用集合 HG_i 表示,元素个数 $\| HG_i \| = \frac{gw_i}{gw_{\min}} \times v_1$,其中 $gw_{\min}$ 为 GW 中的最小权重,并且对于任意 $i \neq j$,有 $HG_i \cap HG_j = \varnothing$。显然 $HG = HG_1 \cup HG_2 \cup \cdots \cup HG_C$,并且 $\| HG \| = v_1 \times \sum_{i=1}^{C} gw_i$。

2) 副本映射函数 ReplicaOnRing(r)。该函数的功能是通过哈希运算,得到 r 在哈希空间环中对应的点。此函数为哈希函数,输入参数 r 为数据对象的标识符;输出用 s 表示,它是数据对象 r 在哈希空间 O 上的映射结果。

3) 副本定位函数 ReplicaInGroup(s,HG,k)。该函数的功能是从 s 出发,顺着哈希空间环顺时针方向找到的 k 个不同分组。输入参数 s 为 LocateOnRing 的返回值,HG 为 GroupOnRing 的返回值,k 为数据对象 r 的副本总数;输出为存储节点分组集合中的 k 个不同分组。

组间副本布局算法具体处理过程如下:

预处理　$GH = h_1(G, v_1, GW)$

这里将 G 中的所有分组映射到 O 上,h_1 为哈希函数,输出给各分组的映射结果。

```
输入　r,k(r 为数据对象,k 为副本数量)。
处理　MapInGroup(r, k)
{
  s = ReplicaOnRing(r);        /* 计算出 r 在 O 中所的值 */
  group = ∅, count = 0; temp = ∅
                /* group 为满足组间副本布局的分组集合 */
  While(count < k)
    {
      temp = ReplicInGroup(s,k);
       /* 沿哈希空间环 O,顺时针方向找到 k 个不同分组 */
      group = group ∪ {temp}; count++;
    }
}
输出　group。
```

1.3 组内副本布局机制

对于数据对象 r_i，其 k 个副本分别为 $\{r_{(i,1)}, r_{(i,2)}, \cdots, r_{(i,k)}\}$，通过组内副本布局算法实现 r_i 的 k 个副本分别映射到 k 个不同的分组。假设在组间副本布局阶段，$\{r_{(i,1)}, r_{(i,2)}, \cdots, r_{(i,k)}\}$ 分别映射到 $\{G_1, G_2, \cdots, G_k\}$，其中 $r_{(i,j)}$ 被映射到分组 $G_j(1 \leqslant j \leqslant k)$ 中，那么在组内副本布局阶段，$r_{(i,j)}$ 将被映射到 G_j 的某个节点中。由于组内各存储节点的性能具有一定差异，因此在组内也引入了虚拟节点，根据组内节点的性能为之分配权重，再依据组内节点权重为节点分配不同数量的虚拟节点，设 NW_j 中最小元素的虚拟节点数 v_2。$r_{(i,j)}$ 在分组 G_j 内部的副本映射过程与组间副本映射过程相似，具体映射过程如下：

预处理 NV_j = NodeOnRing(G_j, v_2, NW_j, O)。

这里通过哈希运算，将 G_j 中的所有存储节点映射到 O 上，映射结果为 NV_j。

输入 r（r 为某数据对象的一个副本）。

处理 MapInNode(r)

```
{
  t = ReplicaOnRing(r);
                    /* 通过哈希运算,返回 r 在 O 上的值 */
  node = ReplicaInNode(t,NV_j);
          /* 沿 O 顺时针方向,找到距离 t 最近的存储节点 */
}
```

输出 $node$。

根据组间和组内副本布局机制可知，每个物理存储节点的虚拟节点数据与该节点的性能成正比，并且所有虚拟节点均地分布在哈希空间中。由 1.1 节定义可知，对于节点 $d(i,j)$，其权重为 $nw(i,j)$，虚拟节点个数为 $\frac{nw(i,j)}{nw(i,min)} \times v_2$，其中 $nw(i,min)$ 表示第 i 个分组中权重最小的节点，所在分组为 G_i，所在分组权重为 gw_i，所在分组的虚拟节点个数为 $\frac{gw_i}{gw_{\min}} \times v_1$，那么对于任意一个数据对象 r，其被分配到节点 $d(i,a)$ 和 $d(i,b)$ 的概率之比为：

$$\frac{P(r \to d(i,a))}{P(r \to d(i,b))} = \frac{(nw(i,a)/nw(i,min) \times v_2) \times (gw_i/gw_{\min} \times v_1)}{(nw(i,b)/nw(i,min) \times v_2) \times (gw_i/gw_{\min} \times v_1)} = \frac{nw(i,a) \times gw_i}{nw(i,b) \times gw_i}$$

在云存储系统中，数据对象的规模非常庞大，映射结果会非常接近理论值，因此 RPGCH 算法是均衡的。

1.4 副本布局动态调整策略

为保证数据副本在各节点之间的均衡分布，当云存储系统中的节点规模发生改变时，需要进行副本迁移，因此，RPGCH 算法需要支持节点规模的变化。节点规模的变化分为两类：1）批量地加入或删除节点。根据云存储系统建设的实际情况，一般批量加入或删除的节点处于同一机架或连接在同一个交换机上，这些节点在同一灾难域中，因此，这种情况可认为是存储节点分组的加入或删除，需进行组内和组间两次副本调整。2）单个节点的加入或删除。在云存储系统的建设过程中很少会出现单节点的加入或删除，这种情况可认为是灾难域内部的节点数量变动，只需进行组内副本调整。

1.4.1 存储节点的批量增加或删除

当批量增加存储节点时，由于每个数据对象有 k 个副本，这 k 个副本分别放置在不同的分组中，并且这 k 个分组在哈希空间中的位置是相对连续的，因此，对于任意的分组 G_i，G_i 中存放有其前面的（$k-1$）个不同分组中的数据副本，并且 G_i 中的数据副本也存放在其后的（$k-1$）个不同分组内。假设 $k = 3$ 批量增加存储节点的副本布局调整过程如图 1 所示。

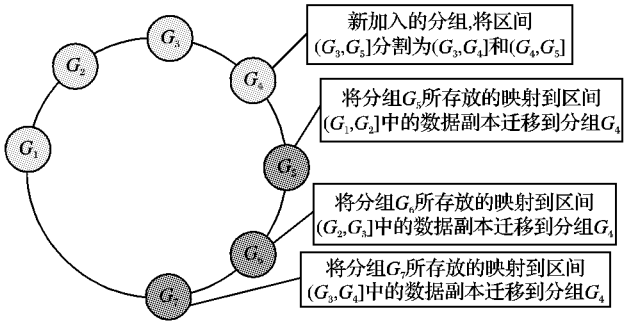

图 1 增加分组的副本布局调整过程

在图 1 中，未加入分组 G_4 之前，对于要放入云存储系统中的任意数据对象 r，如果 $h(r)$ 的值在区间（G_3，G_5］，那么 r 的 3 个副本将分别放置在从 G_5 开始的 3 个不同分组中，即 G_5、G_6、G_7 中。加入分组 G_4 后，系统的副本布局调整过程描述如下：

1）通过哈希算法计算出 G_4 在哈希空间环 O 上的位置；

2）找到 G_4 所在的区间（G_3，G_5］，将区间（G_3，G_5］分为两个区间（G_3，G_4］和（G_4，G_5］；

3）将 G_5 中所存放的映射到区间（G_1，G_2］中的数据对象的副本迁移到分组 G_4 中；

4）将 G_6 中所存放的映射到区间（G_2，G_3］中的数据对象的副本迁移到分组 G_4 中；

5）将 G_7 中所存放的映射到区间（G_3，G_4］中的数据对象的副本迁移到分组 G_4 中；

6）根据组内副本布局算法，将迁移到分组 G_4 中的数据副本映射到具体的存储节点。

对于批量删除存储节点的情况，设删除分组 G_i，则会引起 G_i 的 k 个不同后继分组的副本数量发生变化。设 $k = 3$，批量删除存储节点的副本布局调整过程如图 2 所示。

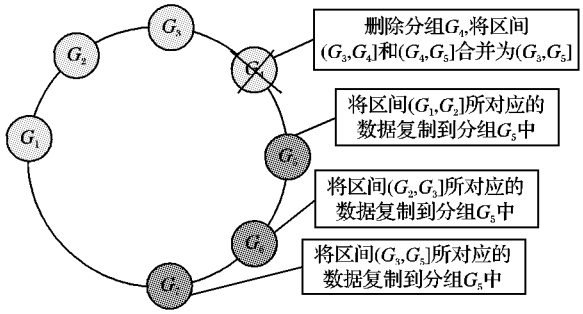

图 2 删除分组的副本布局调整过程

在图 2 中，删除分组 G_4 的副本布局调整过程具体描述如下：

1）将区间（G_3，G_4］和（G_4，G_5］合并为一个区间（G_3，G_5］；

2）对于所有映射到区间（G_1，G_2］的数据对象，在分组 G_5 中创建一个新副本，之后利用组内副本布局算法，将新创建的副本放置到 G_5 的存储节点上；

3）对于所有映射到区间（G_2，G_3］的数据对象，在分组 G_6 中创建一个新副本，之后利用组内副本布局算法，将新创建的副本放置到 G_5 的存储节点上；

4）对于所有映射到区间（G_3，G_4］的数据对象，在分组 G_7 中创建一个新副本，之后利用组内副本布局算法，将新创建的副本放置到 G_5 的存储节点上。

根据批量节点的增删变动情况可知，在批量增加节点时，

副本只从旧节点向新节点迁移,不存在旧节点之间的副本迁移;在批量删除节点时,副本只从被删除节点迁移到剩余节点,不存在剩余节点之间的副本迁移,因此算法是动态自适应的。

1.4.2　单存储节点的增加或删除

单存储节点的增加或删除情况比较简单,直接在节点所加入或删除的分组内部进行调整。虽然单个节点的加入或删除会引起该节点所在分组权重的变化,但由于在云存储系统建设过程中,添加或删除单个节点的情况比较少,若是出现分组内某节点故障,则会立即用新的节点来替换,因此,单节点的添加或删除对系统均衡性的影响会很小。单节点增删的副本布局调整过程与批量节点增删的副本布局调整过程相似,限于篇幅,不再赘述。

2　实验与结果分析

为对 RPGCH 算法的均衡性、动态自适应性、低性能开销进行测试,本文编程实现了 RPGCH 算法。在模拟环境中设置 100 个存储节点,节点的存储性能在 1 ~ 5 随机选取(每个节点的虚拟节点个数与存储性能成正比),将这 100 个节点随机地分为 5 组,每组 20 个,最终得到 5 个分组的存储性能分别为 68,56,62,52,65,利用 MD5 算法将节点和数据副本映射到哈希空间。

2.1　均衡性分析

向系统中的 100 个节点发送 10^6 个数据对象,系统中权重最小分组的虚拟节点个数 $v_1 = 200$,数据对象副本数量 $k = 3$,理论情况下,分配到各组的副本数量与各组的权重成正比。图 3 为组间副本分布结果图,纵轴 $P1$ 表示各分组所分配到的副本数量占系统副本总量的百分比。从图 3 可看出:各分组所分配的副本数量与理论情况相差很小,分组 3 与理论情况的偏差最大,但也在 5% 之内,由此证明 RPGCH 算法在组间副本的放置过程中具有很好的均衡性。

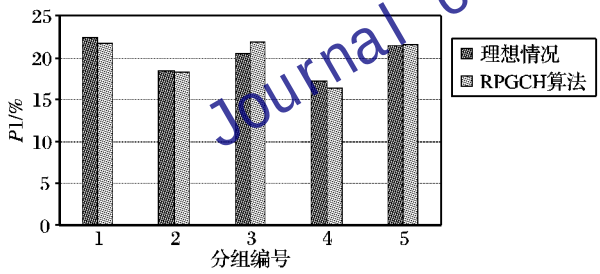

图 3　组间副本布局

图 4 显示了组内副本布局情况,选取第 5 组中的 20 个节点,这 20 个节点的存储性能分别为 1,4,1,3,5,5,5,4,5,3,2,1,4,2,5,5,1,3,1,5,组内权重最小节点的虚拟节点个数 $v_2 = 200$。图 4 中纵轴 $P2$ 表示各节点所分配的副本数量占本分组中副本总量的百分比。

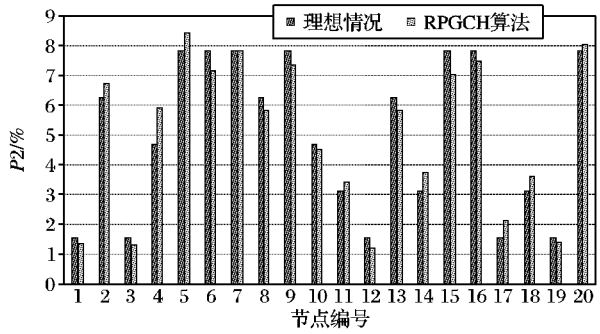

图 4　组内副本布局

从图 4 可看出:各节点所分配的副本数量与理想情况下的偏差都很小,综合图 3 ~ 4 的实验结果可得,RPGCH 算法具有很好的均衡性。

2.2　动态自适应分析

设定数据对象的数量为 10^6 个,依次添加存储性能都为 55 的 3 个分组,再依次删除这 3 个分组,每个分组有 20 个节点,图 5 为组内副本布局随分组增删的变化情况,纵轴 $P3$ 表示迁移的副本数量占副本总量的百分比。设 q 为新增加分组性能占分组总性能的百分比,在理想情况下,数据的迁移量应该与 q 成正比关系,分组的添加不会引起旧分组之间的数据迁移,分组的删除同样如此。从图 5 可看出:当增加或删除分组时,数据的迁移量与理想情况基本相同,所存在的偏差是因为副本在分配过程的不均匀所致。

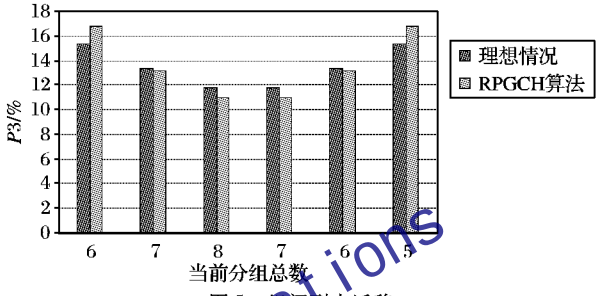

图 5　组间副本迁移

单节点的增删只会引起该节点所在分组内部的数据迁移,选取第 5 个分组,向组内连续 3 次增加存储性能均为 3 的节点,再依次删除这 3 个节点,随着单节点的增删,图 6 显示了组内副本迁移情况,纵轴 $P4$ 表示迁移的副本数量占副本总量的百分比。从图 6 可看出:单节点增删时,算法 RPGCH 同样具有很好的自适应性,迁移的数据量与理想情况下偏差很小。

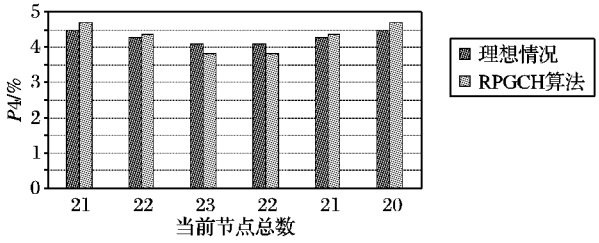

图 6　组内副本迁移

2.3　性能分析

将 10^6 个数据对象映射到系统中,初始时系统中有 10 个节点,之后依次添加 10 个节点,直到节点数量达到 100 个,计算这 10 种情况下单个数据对象的定位时间。图 7 显示了单个数据对象的定位时间随着节点数量增加的变化趋势,从图 7可看出:随着节点数目的增加,数据对象的定位时间变化很小,在节点数目增加的过程中,单数据对象的定位时间都能在几十微秒内完成。在文件系统中,一般用 B^+ 树组织数据对象,如果通过文件系统的 B^+ 树来定位数据对象,那么对于 10^6 的数据规模,其定位时间将远远超过 RPGCH 算法的定位时间。

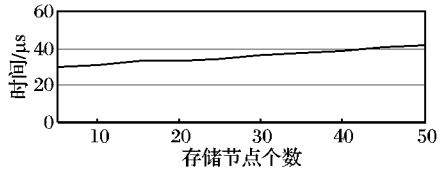

图 7　单副本定位时间变化曲线

3 结语

在具有数千节点的云存储系统中,如何将数据对象均衡、高效地放置到系统中,并能保证数据的高可靠性是一个重大挑战。本文提出的 RPGCH 算法摒弃了记录数据对象映射信息的做法,通过改进的一致性哈希算法获得副本与存储节点之间的映射信息。该算法支持存储节点分组,在每个分组内只存放同一数据对象的一个副本,很大程度上提高了数据的可靠性。此外,该算法分配到每个节点的副本数量与该节点的服务能力成正比,并且能自适应节点数量变化。因此,该算法很适合节点数据庞大的云存储系统,是云存储系统副本布局的一个良好选择。

参考文献:

[1] GRAY J. What next? A few remaining problems in information technology[EB/OL]. [2011-05-10]. http://research. microsoft. com/~gray/ talks/ Gray_Turing_FCRC. pdf.

[2] 云存储[EB/OL]. [2011-05-10]. http://baike. baidu. com/view/2044736. htm.

[3] GHEMAWAT S, GOBIOFF H. The Google file system [C]// Proceedings of the 19th ACM Symposium on Operating Systems Principles. New York: ACM Press, 2003: 19 – 22.

[4] SHVACHKO K, KUANG H, RADIA S. The Hadoop distributed file system [C]// IEEE 26th Symposium on Storage Systems and Technology. Piscataway, NJ: IEEE Press, 2010: 1 – 10.

[5] WEIL S A, BRANDT S A, MILLER E L, *et al.* Ceph: A scalable, high-performance distributed file system [EB/OL]. [2010 - 05-10]. http://www. ssrc. ucsc. edu/Papers/weil-osdi06. pdf.

[6] 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 29th Annual ACM Symposium on Theory of Computing. New York: ACM Press, 1997: 654 – 663.

[7] BRINKMANN A, EFFERT S, auf der HEIDE F M. Dynamic and redundant data placement [C]// ICDCS'07: Proceedings of 27th International Conference on Distributed Computing Systems. Piscataway, NJ: IEEE Press, 2007: 29.

[8] HONICKY R J, MILLER E L. Replication under scalable hashing: a family of algorithms for scalable decentralized data distribution [C]// Proceedings of the 18th International Parallel and Distributed Processing Symposium. Piscataway, NJ: IEEE Press, 2004: 96.

[9] WEIL S A, BRANDT S A, MILLER E L, *et al.* CRUSH: Controlled, scalable and decentralized placement of replicated data [C]// Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. New York: ACM Press, 2006: 31 – 42.

[10] XIAO N, CHEN T. RAEDP: An effective hybrid data placement algorithm for large-scale storage systems [J]. Journal of Supercomputing, 2011, 55(1): 103 – 122.

[11] DeCANDIA G, HASTORUN D. Dynamo: Amazon's highly available key-value store [C]// Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles. New York: ACM Press, 2007: 14 – 17.

(上接第 616 页)

明,在同样的检测时间内准确性优于 Chen 的 NFD-E 失效检测模型和 φ 失效检测模型,并且具有受窗口大小影响较小的特性。该模型能够适应变化的网络环境,可作为失效检测服务使用。

参考文献:

[1] 陈宇江, 魏峻, 杨波, 等. Web 应用服务器的适应性失效检测[J]. 软件学报, 2005, 16(11): 1929 – 1938.

[2] LINDHORST T, LUKAS G, NETT E, *et al.* Data-mining-based link failure detection for wireless mesh networks [C]// Proceedings of the 29th IEEE International Symposium on Reliable Distributed Systems. Piscataway, NJ: IEEE Press, 2010: 353 – 357.

[3] TSAI W, SHAO Q, SUN X, ELSTON J. Real-time service-oriented cloud computing [C]// Proceedings of the 6th World Congress on Services (SERVICES-1). Piscataway, NJ: IEEE Press, 2010: 473 – 478.

[4] GREVE F, SENS P, ARANTES L, *et al.* A failure detector for wireless networks with unknown membership [C]// Proceedings of the 17th International Conference on Parallel Processing. Berlin: Springer-Verlag, 2011, Ⅱ: 27 – 38.

[5] 穆飞, 薛巍, 舒继武, 等. 一种面向大规模副本存储系统的可靠性模型[J]. 计算机研究与发展, 2009, 46(5): 756 – 761.

[6] DING X, HOU Y, GU Z, *et al.* A failure detection model based on message delay prediction [C]// GCC'09: Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing. Washington, DC: IEEE Computer Society, 2009: 24 – 30.

[7] CHEN W, TOUEG S, AGUILERA M K. On the quality of service of failure detectors[J]. IEEE Transactions on Computers, 2002, 51(5): 561 – 580.

[8] HAYASHIBARA N, DEFAGO X, YARED R, *et al.* The φ accrual failure detector [C]// SRDS'04: Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems. Washington, DC: IEEE Computer Society, 2004: 66 – 78.

[9] DÉFAGO X, URBA P, HAYASHIBARA N, *et al.* Definition and specification of accrual failure detectors [C]// DSN'05: Proceedings of the 2005 International Conference on Dependable Systems and Networks. Washington, DC: IEEE Computer Society, 2005: 206 – 215.

[10] BHOLE Y, POPESCU A. Measurement and analysis of HTTP traffic[J]. Journal of Network and Systems Management, 2005, 13(4): 357 – 370.

[11] GOLMIE N, REBALA O. Bluetooth adaptive techniques to mitigate interference [C]// Proceedings of the Global Telecommunications Conference. Piscataway, NJ: IEEE Press, 2003: 405 – 409.

[12] TENG W, CHANG C, CHEN M. Integrating Web caching and Web prefetching in client-side proxies[J]. IEEE Transactions on Parallel and Distributed Systems, 2005, 16(5): 444 – 454.

[13] 张世武, 吴月华, 杨杰, 等. 基于信息寻觅智能体的网络用户浏览模式研究[J]. 计算机研究与发展, 2004, 41(11): 1966 – 1973.

[14] TUSHAR D C, SAM T. Unreliable failure detectors for reliable distributed systems [J]. Journal of the ACM, 1996, 43(2): 225 – 267.

[15] KALEWSK M, KOBUSINSKA A, KOBUSINSKI J. Fast failure detection service for large scale distributed systems [C]// Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing. Piscataway, NJ: IEEE Press, 2009: 229 – 236.