软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(8):1929−1939 [doi: 10.13328/j.cnki.jos.005200]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种基于跳跃 hash 的对象分布算法*

聂世强, 伍卫国, 张兴军, 蔡 毅, 徐志伟

(西安交通大学 电子与信息工程学院,陕西 西安 710049)

通讯作者: 伍卫国, E-mail: wgwu@mail.xjtu.edu.cn

摘 要: 如何有效地将海量数据分布到存储节点,是存储系统首要解决的问题.提出的 MJHAR(matrix-based jump hash algorithm for replication data)对象分布算法简洁、高效,支持权值和数据冗余机制.该算法创造性地将节点映射到二维矩阵,对象的分布、定位只需从矩阵的行内、行间计算目标节点的行号和列号即可.理论研究表明,该算法满足公平性、自适应性、紧凑性、节点变化对象迁移量较小的特点.实验结果表明,该算法的计算时间比一致性 hash 算法快 40%,比跳跃 hash 算法快 23%,极大地缩短了计算时间,且比一致性 hash 算法对象分布更加均匀.

关键词: 数据分布;对象存储系统;跳跃 hash

中图法分类号: TP316

中文引用格式: 聂世强,伍卫国,张兴军,蔡毅,徐志伟.一种基于跳跃 hash 的对象分布算法.软件学报,2017,28(8):1929−1939. http://www.jos.org.cn/1000-9825/5200.htm

英文引用格式: Nie SQ, Wu WG, Zhang XJ, Cai Y, Xu ZW. Object placement algorithm based on jump hash. Ruan Jian Xue Bao/Journal of Software, 2017,28(8):1929−1939 (in Chinese). http://www.jos.org.cn/1000-9825/5200.htm

Object Placement Algorithm Based on Jump Hash

NIE Shi-Qiang, WU Wei-Guo, ZHANG Xing-Jun, CAI Yi, XU Zhi-Wei

(School of the Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: Efficient and scalable object management is a major factor of the overall performance in the object-based storage system. This paper proposes an algorithm MJHAR (matrix-based jump hash algorithm for replication data) that can be applied to the heterogeneous storage system to support variable levels of the object replication. The algorithm runs very fast and moves more minimizing data when storage nodes changes. Experimental results show that the running time of MJHAR is less than that of the consistent hash by 40% and that of jump hash by 23%. The object distribution of MJHAR is more uniform than that of consistent hash algorithm.

Key words: data distribution; object-based storage system; jump hash

随着互联网的不断发展,数据呈现爆炸式的增长.IDC 研究报告指出:到 2020 年,全球数据量将达到 40ZB[1].对象存储系统由于具有块存储系统、文件存储系统无可比拟的可扩展性、数据可靠性和可管理性等特性,已成为当前主流存储架构[2,3].在对象存储系统中,数据以对象的形式存储到对象存储设备(object-based storage device,简称 OSD),对象是读写的基本单位.国内外大量的公司和科研机构,如微软、IBM、谷歌、Oracle、清华大学、中国科学院、华中科技大学,都针对对象存储系统展开了广泛而深入的研究,并且推出了各具特色的存储系统,如 Oracle 的 Ceph 对象存储系统[4]、Facbook 的 Cassandra 存储系统[5]、微软的 Azure 存储系统[6]和中

* 基金项目: 国家重点研发计划(2016YFB1000303); 国家自然科学基金(61672423)

Foundation item: National Key Research and Development Program of China (2016YFB1000303); National Natural Science Foundation of China (61672423)

收稿时间: 2016-06-19; 修改时间: 2016-09-21, 2016-11-11; 采用时间: 2016-12-01; jos 在线出版时间: 2017-01-12

CNKI 网络优先出版: 2017-01-12 10:35:59, http://www.cnki.net/kcms/detail/11.2560.TP.20170112.1035.001.html

国科学院的蓝鲸分布式对象系统[7].

对象存储系统面临着海量对象的管理问题,即对象分布算法如何有效地将对象分布到大量的 OSD 中存储.对象和节点间的映射关系可以抽象为经典装箱问题,装箱问题存在于云计算的存储资源和计算资源的管理[8]等方面.与 GFS 等采用中心化的 master 节点管理数据存储位置的方式不同,目前,研究人员大都采用基于 hash 的算法分布对象到 OSD 节点,保证所有对象映射到存储节点的随机性和存储系统的可扩放性.Karger 等人在 1997 年提出的一致性 hash 算法最初用来解决分布式 Web 服务的缓存问题,随后应用到存储系统中.该算法将所有对象和节点映射到 $0\sim(2^{32})-1$ 空间,对象存储在其顺时针方向最近的节点上,可以使对象大致均匀分布,计算复杂度为 $O(\log n)$[9].但是节点失效后,仅将对象迁移到相邻节点,增加相邻节点负载,且只适用于同构存储系统.其变种算法根据每个节点的权值设置其虚拟节点个数,支持异构系统,但是空间复杂度较高.Sage 等人基于 RUSH 算法提出了 CRUSH 算法,该算法已应用于 Ceph 分布式存储系统,用户能够自定义不同的数据放置规则满足数据可靠性、性能的要求[10].但是该算法节点变化后最多迁移理论值的 h 倍使对象再平衡,其计算时间复杂度是 CRUSH Map 每一层时间复杂度之和.Lamping 等人从降低对象分布的时间复杂度考虑,提出了跳跃 hash 算法[11].该算法可以从理论证明对象均匀分布,时间复杂度为 $O(\log n)$,但是只支持同构系统,不具有多副本策略.谢伟等人针对异构存储系统提出了双模对象分布算法[12].此算法可以使存储系统表现出性能型和容量型模式,最大计算时间为一致性 hash 算法的两倍.Miranda 等人提出了基于动态区间的对象分布算法.该算法将对象映射到(0,1)区间,并将区间划分为多个子区间,每个节点拥有和其权值成正比的多个子区间,以此为依据进行数据映射,但是真实的存储系统节点频繁失效,将会使区间维护变得非常复杂[13].Alon 等人提出了适用于社交网络的 social hash 算法,通用性较差[14].陈涛等人提出了 EFAH hashing 算法[15],可以在集群间和集群内分布对象,但是需要维护区间分配表,应用于真实的存储系统难度较大.

目前,存储系统仍然需要多方面优化,如降低节点间网络开销[16]等,其中,对象分布算法可以在以下两个方面进行改进.

(1) 天气预测、地震预测、电子商务等大数据应用要求存储系统具有高吞吐、低延迟等特性.对象分布算法很大程度影响了存储系统的聚合带宽和处理时间,因此,较快的对象分布算法能够提高存储系统性能.当前的对象分布算法主要着重于对象均匀分布、自定义放置策略等方面.面对大数据应用对存储系统的低延迟、高吞吐的需求,如何降低对象分布算法的计算时间,已成为对象存储系统需要解决的问题.

(2) 对象存储系统中,所有 OSD 节点都会运行包含对象分布算法的程序,因此,简单而高效的对象分布算法能够占用较低内存,降低程序出现 bug 的概率.而当前的对象分布算法有些需要维护表信息,有些处理逻辑复杂,并不适用于真实的存储系统.

本文针对上述问题做了如下工作.

(1) 基于跳跃 hash 算法的思想,提出了一种表示对象副本和节点映射关系的函数,并证明对象映射到每个节点的概率理论上相同.基于此函数思想和动态子数组提出了跳跃查找算法.该算法根据节点的权值计算每个节点的虚拟节点数目,对所有节点的虚拟节点依次分配序号组成数组;同时,每个节点也具有由其虚拟节点组成的子数组,对象的定位、查找过程只需计算对象映射的虚拟节点序号,然后查找此虚拟节点所对应的节点即可.

(2) 将情形(1)中提到的跳跃查找算法和二维矩阵结合,提出了 MJHAR(matrix-based jump hash algorithm for replication data)对象分布算法.该算法初始时将所有节点映射到二维矩阵内.对象的定位查找过程主要分为两步:首先,对象利用跳跃查找算法映射到矩阵的某一行内;其次,在目标行内使用跳跃查找算法映射到矩阵的某一节点.通过理论分析证明:MJHAR 算法是简洁而高效的,对象是均匀分布的,节点变化后对象迁移量是较少的.实验结果表明,MJHAR 算法的计算时间比一致性 hash 算法快 40%,比跳跃 hash 算法快 23%.

本文第 1 节给出描述存储系统和算法相关的概念、术语.第 2 节介绍 MJHAR 算法,并描述理论基础、对象

映射过程、二维矩阵构造过程和节点变化后再平衡过程.第 3 节通过理论分析证明 MJHAR 算法具有均匀分布、迁移量较少等特点,并对算法复杂度进行分析.第 4 节给出实验仿真结果和分析.第 5 节给出本文的结论.

1 问题定义

假定对象数目为 $n(n\geq 1)$,依次编号为 0,1,2,…,$n-1$.每个对象拥有唯一的识别符,用 X_i 表示第 i 个对象的标识符.存储节点数目为 $m(m\geq 1)$,依次编号为 0,1,2,…,$m-1$,并组成数组,其中第 s 个存储节点的权值用 w_s 表示.对象数目理论上无限,且对象数量远远大于节点数量,也就是 $n\geq m$.对象存储系统的对象分布问题可以抽象为数学问题:构造函数 $G(X_i,R,m)=s$,函数 G 表示将对象 X_i 的第 R 个副本映射到第 s 个节点.

对象分布算法的优劣可以用以下 4 个指标来评价[17].

- 公平性:每个节点存储的对象数目和其自身的能力(存储容量、性能和节点带宽等)成正比,即若第 s 个节点存储数据对象数目为 $n\times w_s\Big/\sum_{j=0}^{m-1}w_j$,则该算法是公平的.
- 高效性:若对于任意对象,都可以快速计算得到其存放节点,则该算法是高效的.
- 简洁性:若算法只需要少量信息计算存放对象的节点,且代码实现逻辑简单,则该算法是简洁的.
- 自适应性:存储系统为了满足容量、性能需求会加入新的存储设备,也会替换、删除损坏的设备.此类情况都可以认为是节点的权值发生变化.节点的权值变化后,应进行数据迁移来保证数据的可用性和节点的负载均衡.若迁移后仍是均匀分布的,且迁移过程中尽可能地迁移较少的对象,即对象迁移仅仅发生在权值不变的节点和权值变化的节点之间,权值不变的节点之间不发生对象迁移,则该算法是自适应的.

2 MJHAR 算法

MJHAR 算法的主要思想是:将节点映射到二维矩阵内,从矩阵的行间和行内分别采用跳跃查找算法计算目标节点的行号和列号,行号和列号组合就是目标节点序号.针对对象查找和对象迁移问题,MJHAR 算法与其他算法不同的是,它并不需要复杂的放置规则或者维护、更新表信息等,其逻辑处理极其简单、高效.下面的内容分 3 个部分:理论基础、对象查找过程、构造二维矩阵和对象迁移过程.

2.1 理论基础

跳跃 hash 算法中将节点排序依次编号 0,1,2,…,$m-1$,并将所有对象哈希映射到(0,1)区间,在一维数组内计算对象映射的节点序号[10].本文借鉴跳跃 hash 的思想,提出了表示对象副本与目标节点关系的数列 $p:\{a_m\},\{a_m\}=a_0,a_1,a_2,\ldots,a_m$ 和其递推公式 $H:a_{i+1}=\left\lfloor\dfrac{a_i+1}{r_i}\right\rfloor$.其中,

(1) 数列 P 的项表示对象 X_i 的第 R 个副本映射的节点.数列 P 内的相邻两项中前一项是后一项的上一跳节点,后一项是前一项的下一跳节点.如数列 P:0,2,5,……,对象 X_i 的第 R 个副本可以映射到 0 号、2 号、5 号节点.0 号节点是对象 X_i 的第 R 个副本的初始存放的节点.2 号节点是 0 号节点的下一跳节点,并且是 5 号节点的上一跳节点.

(2) 递推公式 H 表示数列 P 的项,即存储对象 X_i 的第 R 个副本的上一跳节点和下一跳节点之间满足的关系,初始时 $a_0=0$.

(3) 递推公式 H 中,r_i 表示以对象 X_i 的第 R 个副本为种子产生(0,1)区间的随机数序列,且产生的随机数在(0,1)区间内均匀分布.

如表 1 所示,举例说明存储对象 X_1,X_2,X_3 的副本与节点关系的数列.

因为存储系统中节点数目是有限的,数列 P 引入限制条件,即对象 X_i 的第 R 个副本的数列 P 最后一项 a_i 作为其最终存储的节点,且满足 $a_i<m\leq a_{i+1}$.若 $m=10$,则表 1 中存储对象 X_1 第 1 个副本的节点、对象 X_2 第 3 个副本和对象 X_3 的第 2 个副本的节点分别为第 9 号节点、第 8 号节点和第 7 号节点.以上内容可得对象 X_i 第 R 个

副本与映射节点关系的跳跃查找函数 $G(X_i,R,m)=a_i(a_i\in P,a_i<m\le a_{i+1},r_i\in hash(X_i,R))$.该函数的证明过程见第 3.1 节.

Table 1 Mapping OSD No. of the object replicas X_1, X_2, X_3

表 **1** 对象 X_1,X_2,X_3 副本映射的 OSD 编号

X_1, R=1	0	4	7	9	13	15	17	67	100
X_2, R=3	0	2	3	5	6	7	8	10	30
X_3, R=2	0	1	4	5	6	7	13	15	17

2.2 数据定位过程

MJHAR 算法支持权值和数据冗余机制.计算对象 X_i 的第 R 个副本的存储节点的伪码如图 1 所示,主要分为 2 步:(1) 假设矩阵 M 的行是具有较大权值的虚拟节点,且本行所有节点之和 $\sum_{j=0}^{k}M_{ij}$ 为矩阵 M 第 i 行的权值 RW_i,采用跳跃查找算法在矩阵 M 中计算对象 X_i 第 R 个副本映射的目标行;(2) 目标行内采用跳跃查找算法计算映射的节点.

```
select(x,r,M)
   row_sum= ∑_{j=0}^{h-1} RW_j
   row=jump(x,r,row_sum)
   column_sum= ∑_{i=0}^{k-1} M_{row_i}
   column=jump(x,r,column_sum)
return row*k+column
```

Fig.1 Object distribution alogrithm

图 1 对象分布算法

MJHAR 算法的目标行和行内目标节点的计算都采用跳跃查找算法,跳跃查找算法将函数 G 的思想和动态子数组相结合,首先根据节点的权值为每个节点分配相应数目的虚拟节点,若节点的权值为 5,则该节点具有 5 个虚拟节点.将所有节点的虚拟节点排序分配序号并组成数组 $L=0,1,2,...,\sum_{i=0}^{m-1}w_i$, 在数组 L 内,第 i 个节点的虚拟节点组成的子数组 $L_i=\left[\sum_{j=0}^{i}w_j,\sum_{j=0}^{i+1}w_j\right]$. 首先计算对象的副本映射的虚拟节点序号;然后,通过查表法确定虚拟节点所对应的节点,其具体实现算法如图 2 所示.

```
total_node= ∑_{j=0}^{i} w_j
jump(x,r,total_node)
   b=-1
   j=0
   random.seed(x,r)
   while (j<total_node)
      {
         b=j
         g=random()
         j=floor((b+1)/g)
      }
   id=look_up_table(b)
   return id
```

Fig.2 Skiped-Searched alogrithm

图 2 跳跃查找算法

存储系统为了提高性能、降低程序出现 bug 概率,其每一模块理论设计应该精简高效,代码逻辑简单明了.MJHAR 算法相对于其他对象分布算法具有两个方面的优势.

- 对象采用概率理论映射到节点,此过程仅涉及计算操作和少量查表操作;而动态区间映射算法需要不断维护每个节点拥有的区间段,存储系统的扩容、删除节点等操作会造成节点拥有的区间段碎片化,

其维护将会变得复杂.

- 本算法可以在百行代码内实现,逻辑清晰;而 Crush 算法等需要数千行代码实现.

因此可见,MJHAR 算法满足高效性和简洁性,可以有效避免 Bug 的出现,提高程序运行的可靠性.

2.3 构造二维矩阵及对象迁移过程

MJHAR 算法将所有节点映射到二维矩阵 M 内,采用跳跃查找算法分别计算对象的目标节点在二维矩阵的行号和列号.为了使行、列查找更加公平,根据节点个数 m 构造二维加权矩阵 $M[h][k]$,其中,$\sqrt{m}\le h,k\le\sqrt{m}+1$.存储节点和二维加权矩阵 M 的映射关系为:第 s 个节点对应二维加权矩阵的第 s/k 行、第 $s\%k$ 列,并且 $M[s/k][s\%k]$的值是第 s 个节点的权值 w_s.节点数 m 不一定等于二维加权矩阵 M 的行列之积,因此,二维矩阵 M 从 $M[(m-1)/k][(m-1)\%k+1]$到 $M[h-1][k-1]$的值设为 0.以上为二维加权矩阵 $M[h][k]$的构造过程.

存储设备的增加、删除和更换都会造成节点的权值发生变化.本算法针对此种情况简单易行.若是加入节点,按照行遍历矩阵 M,若查找到元素值为 0,即为空余位置,则新加入的存储节点放入空余位置.如果矩阵 M 已满,则需要扩容矩阵 M:为了保证矩阵 M 的行列满足关系 $\sqrt{m}\le h,k\le\sqrt{m}+1$,首先判断矩阵的行、列关系,若矩阵 M 行大于列,则矩阵 M 新增一列;若矩阵 M 列大于行,则矩阵 M 新增一行.若是删除节点,则该节点在矩阵 M 中相应元素的值为 0.若是替换节点,则矩阵 M 中相应元素的值为替换后的权值.节点的权值变化后对所有对象重新计算其目标节点,若对象的存放节点发生变化,则迁移其至新目标节点.

3 算法分析

3.1 公平性证明

第 2.1 节中描述了表示对象副本和节点映射关系的函数 G,本节证明对象 X_i 的第 R 个副本映射到节点 0,1,2,…,$m-1$ 的概率相等.首先推导数列 P 的通项公式.递推公式 $H:a_{i+1}=\left\lfloor\frac{a_i+1}{r_i}\right\rfloor$ 中包含向下取整运算,且 $0<r_i<1$.根据节点数目 m,可以将 r_i 所在的(0,1)区间划分为 $m-a_i$ 份,即 $\left(0,\frac{a_i+1}{m}\right),\left(\frac{a_i+1}{m},\frac{a_i+1}{m-1}\right),\ldots,\left(\frac{a_i+1}{a_i+2},\frac{a_i+1}{a_i+1}\right)$,则递推公式共有 $m-a_i$ 种可能性.第 j 个区间对应的递推公式:

$$H:a_{i+1}=m-j+1,2\le j\le m-a_i,$$

$$H:a_{i+1}\ge m,j=1.$$

若对象 X_i 的第 R 个副本的数列 P 的最后一项是 $a_i=k(k<m)$,则对象 X_i 的第 R 个副本最终映射到节点 k 的概率 $p(a_i=k)$等于 $a_{i-1}<k$,即上一跳节点分别是 0,1,2,……,$k-1$ 时,其下一跳节点是 k 的概率之和与节点 k 下一跳节点序号大于 m,即 $a_{i+1}>m$ 的概率之积;且 $a_{i-1}\in\{0,1,2,\ldots,k-1\}$时,$a_i$ 等于 k 的概率是 $p(a_i=k|a_{i-1}=j)\cdot p(a_{i-1}=j),0\le j\le k-1$.因此,只需递归计算 $a_{i-1}\in\{0,1,2,\ldots,k-1\}$的概率 $p(a_{i-1})$.此处省略递归计算过程,最终计算得到 $a_{i-1}\in\{0,1,2,\ldots,k-1\}$时,下一跳节点是 $a_i=k$ 的概率之和是 $1/(k+1)$.$a_{i+1}>m$ 的概率等于 $p(a_{i+1}=l|a_i=k)\cdot p(a_i=k)=a_i+1/m=(k+1)/m,l>m$.可得对象 X_i 的第 R 个副本存放到节点 k 的概率 $p(a_i=k)=1/m$.因此,函数 G 中对象的副本映射到每个节点的概率相等.

由上文可得:跳跃查找算法中,数组 L 内,每个虚拟节点存储任意对象副本的概率是 $1/\sum_{s=0}^{m-1}w_s$.因此,每个虚拟节点存储对象数目的期望值为 $n\times1/\sum_{s=0}^{m-1}w_s$.因第 s 个节点的虚拟节点组成的子数组 $L_s=\left[\sum_{j=0}^{s}w_j,\sum_{j=0}^{s+1}w_j\right]$,所以第 s 个节点存放的对象数目理论值是 $n\times w_s/\sum_{s=0}^{m-1}w_s$,与其权值成正比.因此,跳跃查找算法是均匀分布的.

MJHAR 算法分别从行内和行间调用跳跃查找算法.第 1 次以行为单位计算目标行过程中,第 i 行被选中的概率是 $\sum_{j=0}^{k-1}M_{ij}/\sum_{i=0}^{h-1}\sum_{j=0}^{k-1}M_{ij}$;第 2 次在目标行计算目标节点过程中,若第 i 行为目标行,矩阵 M 的第 i 行、第 j 列节点被选中的概率是 $M_{ij}/\sum_{j=0}^{k-1}M_{ij}$.由此可得,矩阵 M 中,第 i 行、第 j 列的节点被选中的概率是

$$\sum_{j=0}^{k-1}M_{ij}\Big/\sum_{i=0}^{h-1}\sum_{j=0}^{k-1}M_{ij}\times M_{ij}\Big/\sum_{j=0}^{k-1}M_{ij}=M_{ij}\Big/\sum_{i=0}^{h-1}\sum_{j=0}^{k-1}M_{ij}.$$

节点存储的对象数目与其权值成正比,因此证明 MJHAR 算法是公平的.

3.2 迁移量分析

由第 3.1 节的证明可知:无论跳跃查找算法中节点的权值是否变化,任意对象的副本映射到每个虚拟节点概率相等.因此,权值变化只会影响每个节点存储对象数目的变化.而节点存储对象数目与其权值成正比的关系不会受到影响,因此跳跃查找算法和 MJHAR 算法可以保证节点变化后对象仍然均匀分布.

- 节点加入、权值增加的情况

权值 w_m 的节点加入存储系统,可以认为分 w_m 次加入权值为 1 的存储节点,因此,首先分析加入权值为 1 的节点数据量迁移量.第 3.1 节的证明中,节点数目增加,区间数从 $m-a_i$ 个增加到 $m+1-a_i$ 个,则每个区间的范围都将减少.可以计算出:节点加入后,当前所有节点会迁移其自身 $1\Big/\left(\sum_{s=0}^{m-1}w_s+1\right)$ 的数据对象到新节点,则前 m 个节点迁移数据量之和是 $n\times\sum_{s=0}^{m-1}\left(w_s\Big/\sum_{i=0}^{m-1}w_i\times 1\Big/\left(\sum_{i=0}^{m-1}w_i+1\right)\right)=n\times 1\Big/\left(\sum_{s=0}^{m-1}w_s+1\right)$. 跳跃查找算法再平衡后,新节点拥有对象数目为 $n\times 1\Big/\left(\sum_{s=0}^{m-1}w_s+1\right)$. 同理可证:权值为 w_m 的节点加入存储系统后将分配 $n\times w_m\Big/\sum_{s=0}^{m}w_s$ 个对象,且前 m 个节点的数据迁移量之和是 $n\times w_m\Big/\sum_{s=0}^{m}w_s$, 因此可以得出,新节点分配的对象数目和其他节点迁移量之和相等.即前 m 个节点之间并未发生数据迁移.因此,跳跃查找算法是数据迁移量最优的.MJHAR 算法由于调用两次跳跃查找算法,会有两次对象迁移.在第 1 次数据迁移中数据迁移最优,但是第 2 次数据迁移会导致行内节点发生不必要的对象迁移.因此,节点的权值增加或加入新节点,MJHAR 算法对象迁移量是较优的.

- 节点删除、权值减少的情况

因为跳跃查找算法中对象 X_i 存放的节点和节点在数组 L 的位置有关,所以存储系统的节点失效后,对象会依次迁移到后续节点.即第 s 个节点失效后,第 s 个节点的所有对象迁移到第 $s+1$ 个节点,第 $s+1$ 个节点的所有对象依次迁移到第 $s+2$ 节点,第 $m-2$ 个节点的所有对象迁移到第 $m-1$ 个节点.第 s 个节点的权值变化后,最大对象迁移量为 $\sum_{k=s+1}^{m-1}\left(n\times w_k\Big/\sum_{i=0}^{m-1}w_i\right)+n\times\Delta w\Big/\sum_{i=0}^{m-1}w_i$, Δw 是第 i 个节点的权值变化前后之差.此公式表明了对象迁移量和失效节点在数组 L 位置的关系.失效节点越处于存储节点数组 L 后端,对象迁移量越少.当失效节点是第 $m-1$ 个节点时,对象迁移量最优;当失效节点是第 1 个节点时,会造成所有对象重新分布.MJHAR 算法从两个方面避免了跳跃查找算法节点失效后,对象大量重分布的问题:首先,第 1 次以行为单位的计算过程中,因为很少或者不会出现某一行节点集体失效的情况,所以行间不会产生对象大规模迁移;其次,对于矩阵 M 行内某一节点的失效的情况,即使是行内首节点失效,对象大规模迁移也仅仅会发生行内的节点,不会导致除本行之外的节点对象发生大规模迁移.因此,对于任意节点的权值变化,MJHAR 算法能够保证较少的对象迁移量.

综上所述,可以得出 MJHAR 算法是自适应的这一结论.

3.3 性能分析

第 3.1 节中提到:当 $a_{i-1}=0,1,2,\ldots,k-1$ 时,$a_i=k$ 的概率之和是 $1/(k+1)$,因此对象副本的映射节点序号的期望值是 $1/2+1/3+\ldots+1/m\approx\ln m$,跳跃查找算法中,数组 L 的长度等于所有节点的子数组长度之和,节点子数组长度为常数,因此,其计算复杂度为 $O(\log m)$.跳跃查找算法仅仅需要存储每个节点的信息(ip 地址和权值等),其空间复杂度为 $O(m)$.

MJHAR 算法的计算时间包括矩阵 M 的构造时间和对象的计算时间.由于矩阵 M 的构造过程只需 1 次,随着对象定位的次数增加,其构造时间可以被无限次的对象计算过程分摊,因此构造时间可以忽略.主要考虑对象计算过程花费的时间.MJHAR 算法调用两次跳跃查找算法.矩阵 M 的行、列长度为 $\sqrt{m}$ 或 $\sqrt{m}+1$,因此,MJHAR 算法的时间复杂度为 $2\times O\left(\log\sqrt{m}\right)=O(\log m)$. MJHAR 算法仅仅存储节点的信息(ip 地址和权值等),其空间复杂度为 $O(m)$.这里,虽然 MJHAR 算法的时间复杂度为 $O(\log m)$,但是其计算时间 $\ln m$ 和 $\log m$ 相差常量 $\log_2 e$.

4 实验仿真

本文在模拟环境下比较 MJHAR 算法、一致性 hash 算法和跳跃 hash 算法的计算时间、对象分布均匀性和对象迁移量这 3 个方面.首先介绍模拟实验的环境:本实验操作系统为 ubuntu 14.04 LTS 系统,采用 python 语言实现 MJHAR 分布算法.一致性 hash 算法和跳跃 hash 算法源码从 python 官方仓库[18]下载所得.仿真程序首先随机生成 32 位字符串表示对象的标识符,随后分别调用 3 种对象分布算法计算存储对象的 OSD 节点编号.最后统计每个 OSD 节点的对象数目.通过比较某个 OSD 节点的权值改变前后,每个节点拥有的对象变化量来模拟节点的增加和失效后对象迁移量.以下图表数据来自于对不同节点数目组成的存储系统模拟读写 30 万个对象的实验结果.节点数目以 30 个节点为步长,从 50 个节点到 620 个节点递增.因为跳跃 hash 算法不支持权值,当前所有节点的权值设为 1.

如图 3 所示为 MJHAR 算法、跳跃 hash 算法和一致性 hash 算法的计算时间.从图中可以看出,MJHAR 算法是最快的,跳跃 hash 算法次之,一致性 hash 算法的计算时间最长.虽然跳跃 hash 算法、MJHAR 算法和一致性 hash 算法时间复杂度都为 $O(\log n)$,但是 MJHAR 算法和跳跃 hash 算法其迭代次数期望值为 $\ln n$,其计算时间和一致性 hash 算法有常数 $\log_2 e$ 的差别,因此,MJHAR 算法和跳跃 hash 算法快于一致性 hash 算法.MJHAR 算法调用两次跳跃查找算法,且每一次的迭代次数期望值都小于 $\ln\sqrt{n}$,则两次之和更小于 $\ln n$,所以 MJHAR 算法比跳跃 hash 算法计算得更快.

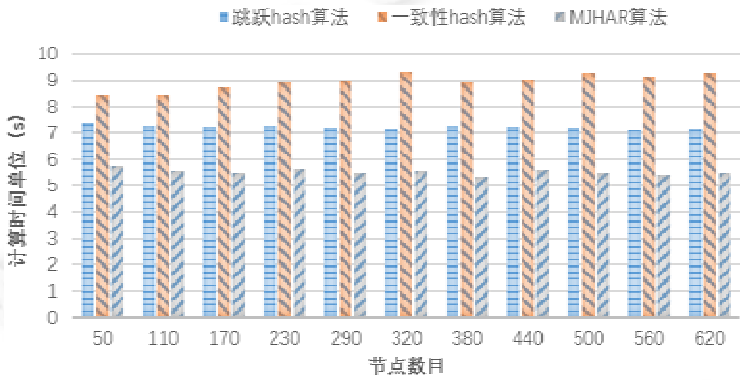

Fig.3 Computation time of mapping the object into OSDs

图 3 对象计算时间比较

图 4 是 3 种算法的均匀性比较.横坐标表示节点数目,纵坐标表示存储系统中节点存储对象数目的标准方差.从图中可以看到,MJHAR 算法和跳跃 hash 算法的方差比一致性 hash 算法的方差小.即这两种算法对象分布得更加均匀,有效地保证了存储节点的负载均衡.因为 MJHAR 算法和跳跃 hash 算法中每个节点分配的对象数目与其权值成正比,而一致性 hash 算法只是将节点、对象随机分布到 $0\sim(2^{32})-1$ 空间,对象存储到顺时针最近的节点,当节点和对象较多时,可以保证数据对象大致均匀分布,因此,MJHAR 算法和跳跃 hash 算法的对象分布更加均匀.

本文模拟比较了 3 种算法节点的权值变化后的对象迁移量.第 3.2 节提及 MJHAR 算法对象迁移量与节点位置有关.为了不失一般性,本文模拟比较了中间节点变化后,3 种算法的对象迁移量.图 5~图 8 展示了 50 个节点、320 个节点组成的存储系统中节点增加、删除后的对象迁移量.图 5、图 6 是模拟 50 个和 320 个节点组成的存储系统,其第 25 号、第 160 号节点失效后每个节点的对象迁移量.从图中可以看出,一致性 hash 算法对象迁移量较少,跳跃 hash 算法和 MJHAR 算法的前一半节点对象迁移量也较少,但是后一半节点对象迁移量较大.且跳跃 hash 算法后一半节点的对象迁移量是 MJHAR 算法的 4 倍左右.

如图 7、图 8 所示是模拟 50 个和 320 个节点组成的存储系统,第 25 号、第 160 号位置增加节点后每个节

点的对象迁移量.从图中同样可以看出,一致性 hash 算法对象迁移量较少,跳跃 hash 算法和 MJHAR 算法的前一半节点对象迁移量也较少,但是后一半节点对象迁移量较大.且跳跃 hash 算法的后一半节点对象迁移量是 MJHAR 算法的对象迁移量 5 倍左右.

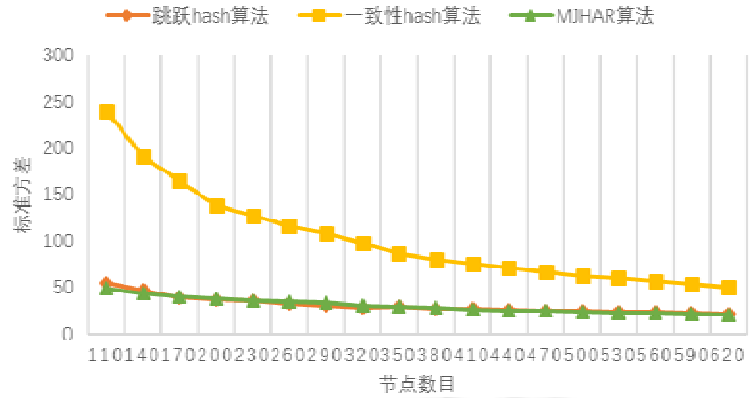

Fig.4 Compare the diviation of data distribution

图 4 对象分布均匀性比较

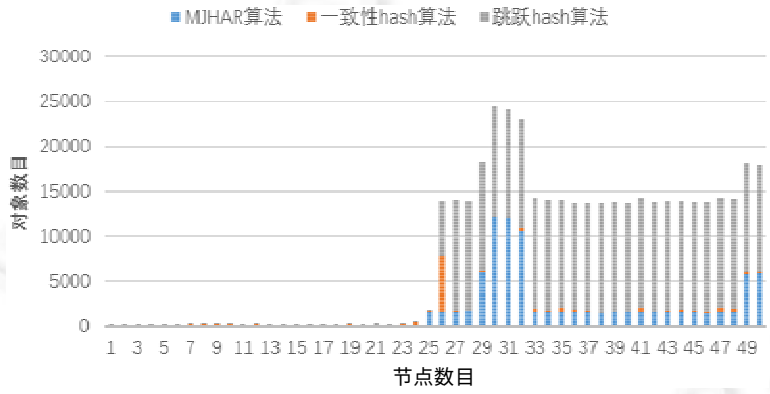

Fig.5 Movements of data between OSDs under 50 OSDs after the 25th OSD failed

图 5 50 个节点下,第 25 号节点失效后的对象迁移量

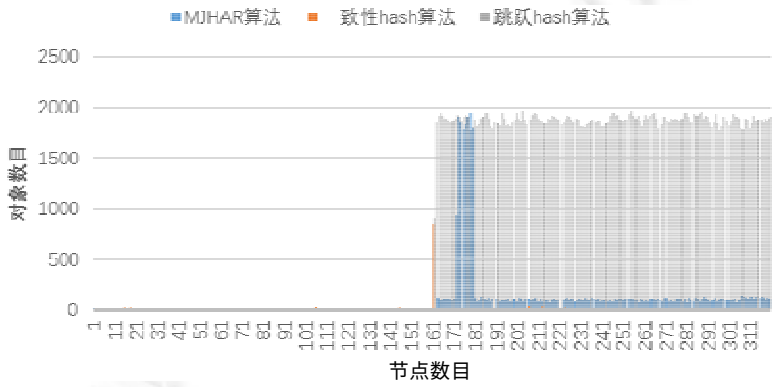

Fig.6 Movements of data between OSDs under 320 OSDs after the 160th OSD failed

图 6 320 个节点下,第 160 号节点失效后的对象迁移量

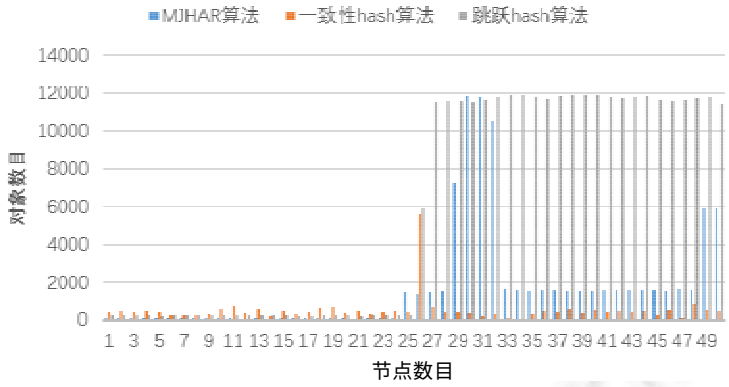

Fig.7 Movements of data between OSDs under 50 OSDs after insert an OSD before the 25th OSD

图 7 50 个节点下,在第 25 号节点前插入节点后的对象迁移量

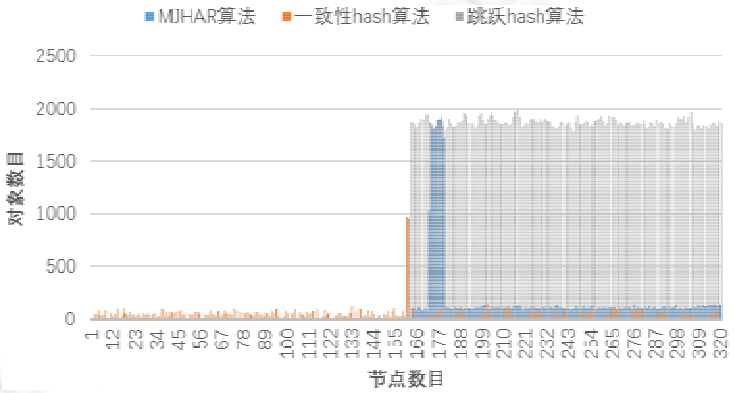

Fig.8 Movements of data between OSDs under 50 OSDs after insert an OSD before the 160th OSD

图 8 320 个节点下,在第 160 号节点前插入节点后的对象迁移量

从图 5~图 8 中可以看出,在节点增加、删除的情况下,一致性 hash 算法对象迁移量最少,但是每个节点的对象迁移量是非均匀的.跳跃 hash 算法和 MJHAR 算法对象迁移量较大,节点对象迁移量是相对均匀的,并且 MJHAR 算法对象迁移量比跳跃 hash 算法数据迁移量更少.

但值得注意的是,一致性 hash 算法是在节点删除后将其拥有的对象迁移到相邻节点或节点增加后从相邻节点迁移对象到新加入节点,因此节点的变化仅仅影响到相邻节点,也就是加大了或者减轻了相邻节点的负载;而 MJHAR 算法和跳跃 hash 算法是将删除节点的对象迁移到所有剩余节点或者从所有节点迁移对象到新加入节点,保证再平衡后对象均匀分布,因此系统不会出现负载不均的现象.

MJHAR 算法一方面相对于跳跃 hash 算法,其对象迁移量降低了数倍;另一方面,相对于一致性 hash 算法的对象迁移量还是较大.这是因为 MJHAR 算法行间对象迁移量较小,但是权值变化的节点,其所在行内的对象迁移量还是较大.我们将在后续研究中降低行内对象迁移量.

5 结 论

大数据时代下,对象分布算法对对象存储系统的性能影响更加显著.目前,大部分对象分布算法计算时间较长,实现逻辑复杂,很难适用于存储系统.

本文提出了一种高效而简洁的对象分布算法 MJHAR.与其他对象分布算法将节点映射在一维数组的方式不同,该算法创造性地将节点映射到二维矩阵内,对象的分布、定位只需从矩阵的行内、行间计算目标节点的

行号和列号即可,降低了对象分布时间.该算法支持权值和数据冗余策略,同时具有公平性、简洁性、自适应性的特点.

实验结果表明:

- MJHAR 算法在实际计算时间上,相对于跳跃 hash 算法和一致性 hash 算法分别下降了 23%和 40%.
- MJHAR 算法相对于一致性 hash 算法分布更加均匀,更加接近理论最优值.
- 相对于跳跃 hash 算法,节点变化后,MJHAR 算法对象迁移量较少.

因此,MJHAR 算法更加适用于异构对象存储系统.

References:

[1] IDC. IDC Digital Universe Study: Big Data, Bigger Digital Shadows and Biggest Growth in the Far East Methodology, Firsts and History of IDC Digital Universe Study. 2012. 1−13.

[2] Azagury A, Dreizin V, Factor M, Henis E, Naor D, Rinetzky N. Towards an object store. In: Proc. of the 20th IEEE/11th NASA Goddard Conf. on Mass Storage Systems and Technologies. 2003. 165−176. [doi: 10.1109/MASS.2003.1194853]

[3] Factor M, Meth K, Naor D, Rodeh O, Satran J. Object storage: The future building block for storage systems a position paper. In: Proc. of the 2005 IEEE Int'l Symp. on Mass Storage Systems and Technology. 2005. 119−123. [doi: 10.1109/LGDI.2005.1612479]

[4] Weil SA, Brandt SA, Miller EL, Long DDE, Maltzahn C. Ceph: A scalable, high-performance distributed file system. In: Proc. of the 7th Symp. on Operating Systems Design and Implementation (OSDI 2006). 2006. 307−320.

[5] Lakshman A, Malik P. Cassandra: A decentralized structured storage system. ACM SIGOPS Operating Systems Review, 2010, 44(2):35−40. [doi: 10.1145/1773912.1773922]

[6] Calder B, Simitci H, Haridas J, Uddaraju C, Khatri H, Edwards A. Windows azure storage: A highly available cloud storage service with strong consistency. In: Proc. of the 23rd ACM Symp. on Operating Systems Principles (SOSP 2011). 2011. 143−157. [doi: 10.1145/2043556.2043571]

[7] Yang DZ, Huang H, Zhang JG, Xu L. BWFS: A distributed file system with large capacity, high throughput and high scalability. Journal of Computer Research and Development, 2005,42(6):1028−1033 (in Chinese with English abstract). [doi: 10.1360/crad20050619]

[8] Luo G, Qian Z, Dong M, Ota K, Lu S. Network-Aware re-scheduling: Towards improving network performance of virtual machines in a data center. In: Proc. of the Int'l Conf. on Algorithms and Architectures for Parallel Processing. Berlin: Springer Int'l Publishing, 2014. 255−269. [doi: 10.1007/978-3-319-11197-1_20]

[9] Karger D, Leightonl T, Lewinl D, Lehman E, Leighton T, Panigrahy R. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In: Proc. of the 29th Annual ACM Symp. on Theory Computing (STOC'97). 1997. 654−663. [doi: 10.1145/258533.258660]

[10] Weil SA, Brandt SA, Miller EL, Maltzahn C. CRUSH: Controlled, scalable, decentralized placement of replicated data. In: Proc. of the 2006 ACM/IEEE Conf. on Supercomputing. New York: ACM Press, 2006. [doi: 10.1109/SC.2006.19]

[11] Lamping J, Veach E. A fast, minimal memory, consistent hash algorithm. arXiv Prepr arXiv14062294, 2014.

[12] Xie W, Zhou J, Reyes M, Noble J, Chen Y. Two-Mode data distribution scheme for heterogeneous storage in data centers. In: Proc. of the 2015 IEEE Int'l Conf. on Big Data (Big Data). IEEE, 2015. 327−332. [doi: 10.1109/BigData.2015.7363772]

[13] Miranda A, Effert S, Kang Y, Miller EL, Popov I, Brinkmann A. Random slicing: Efficient and scalable data placement for large-scale storage systems. ACM Trans. on Storage, 2014,10(3):1−35. [doi: 10.1145/2632230]

[14] Shalita A, Karrer B, Kabiljo I, Sharma A, Presta A, Adcock A. Social hash: An assignment framework for optimizing distributed systems operations on social networks. In: Proc. of the 13th USENIX Symp. on Networked Systems Design and Implementation. 2016. 455−468.

[15] Chen T, Xiao N, Liu F. An efficient hierarchical object placement algorithm for object storage systems. Journal of Computer Research and Development, 2012,49(4):887−899 (in Chinese with English abstract).

[16] Wang F, Qian Z, Zhang S, Dong M, Lu S. SmartRep: Reducing flow completion times with minimal replication in data centers. In: Proc. of the 2015 IEEE Int'l Conf. on Communications (ICC). IEEE, 2015. 460−465. [doi: 10.1109/ICC.2015.7248364]

[17] Brinkmann A, Salzwedel K, Scheideler C. Compact, adaptive placement schemes for non-uniform requirements. In: Proc. of the 14th Annual ACM Symp. on Parallel Algorithms and Architectures (SPAA 2002). 2002. 53−62. [doi: 10.1145/564870.564878]

[18] The python package index. 2017. https://pypi.org/

附中文参考文献:

[7] 杨德志,黄华,张建刚,许鲁.大容量、高性能、高扩展能力的蓝鲸分布式文件系统.计算机研究与发展,2005,42(6):1028−1033. [doi: 10.1360/crad20050619]

[15] 陈涛,肖侬,刘芳.对象存储系统中一种高效的分层对象布局算法.计算机研究与发展,2012,49(4):887−899.

聂世强(1993－),男,河南信阳人,博士生,主要研究领域为高性能存储,云存储.

伍卫国(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为计算机体系结构,云计算,嵌入式系统.

张兴军(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高性能计算机体系结构,网络存储,网络计算.

蔡毅(1994－),男,硕士生,主要研究领域为分布式存储.

徐志伟(1981－),男,博士生,主要研究领域为云存储,高性能存储.