第52卷 第6期 西 安 交 通 大 学 学 报 Vol. 52 No. 6
2018年6月 JOURNAL OF XI'AN JIAOTONG UNIVERSITY Jun. 2018

DOI：10.7652/xjtuxb201806005

采用弧映射的双层对象分布算法

聂世强，伍卫国，崔金华，薛尚山，刘钊华
（西安交通大学电子与信息工程学院，710049，西安）

摘要：针对由混合存储设备组成的对象存储系统中数据对象分布时间开销过大的问题，提出了支持权值和副本机制的弧映射双层对象分布(TMHR)算法。该算法利用存储系统批量扩容、删除的特点，将存储系统内存储节点划分为多个子集群，采用可扩展的子集群哈希(SHFC)算法将数据对象按概率分布到子集群上以保证分布的公平性；在子集群内采用随机置换算法将数据对象等概率分布到节点上以降低数据对象分布时间开销。实验结果表明：与一致性哈希算法和随机切片算法相比，TMHR算法的数据对象分布时间分别缩短了20%和28%；数据对象的分布也更加接近理论情况；在存储节点数变化后，存储系统能够迁移较少的数据对象以进行重新均衡。该算法满足公平性、高效性、简洁性和自适应性，可以降低存储系统I/O路径的时延，提高存储系统性能，较适用于异构混合对象存储系统。

关键词：异构存储；对象存储系统；对象分布算法；公平性；自适应性
中图分类号：TP391 **文献标志码**：A **文章编号**：0253-987X(2018)06-0030-06

An Object Distribution Algorithm Based on Arc and Two-Layer Mapping

NIE Shiqiang, WU Weiguo, CUI Jinhua, XUE Shangshan, LIU Zhaohua
(School of the Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, Chian)

Abstract: An object distribution algorithm based on arc and two-layer mapping is proposed to solve the problem of too much cost time of object distribution in object storage systems consisting of hybrid storage nodes. The algorithm supports weighted allocation and variable levels of object replication mechanism and uses the characteristics of storage system expansion and deletion in batch to divide the storage nodes in the storage system into multiple sub-clusters. The extensible subject group hashing (SHFC) algorithm and a random replacement algorithm are applied to distribute data objects to node to ensure the uniformity of the distribution and to reduce the time overhead. Theoretical analysis shows that the proposed algorithm is fair, simple, effective and adaptive. Experimental results and comparisons with the consistent hashing algorithm and the random slicing algorithm show that the running time of the proposed algorithm reduces 20% and 28%, respectively, and the object distribution of the algorithm is also closer to the theoretical situation. The storage system migrates a minimized number of objects to rebalance the distribution when the number of storage nodes changes. It is concluded that the algorithm could reduce storage system I/O path latency and improve storage system I/O performance, so it is relatively suitable for heterogeneous hybrid object storage system.

收稿日期：2018-01-11。 **作者简介**：聂世强(1993—)，男，博士生；伍卫国(通信作者)，男，教授，博士生导师。 **基金项目**：国家重点研究发展计划资助项目(2016YFB1000303)；国家自然科学基金资助项目(91630206)；海外及港澳学者合作研究基金资助项目(61628210)。
网络出版时间：2018-03-27 **网络出版地址**：http://kns.cnki.net/kcms/detail/61.1069.T.20180327.0847.014.html

Keywords: heterogeneous storage; object-based storage system; object placement algorithm; fairness; adaptability

对象存储系统由于可扩展和易管理等特点，已经发展成为主流存储系统架构。常见的对象存储系统有 Red Hat 公司的 Ceph 存储系统[1]、Facebook 的 Cassandra 存储系统[2]等。对象存储系统中对象是读写的基本单位。客户端调用对象分布算法将对象分布到对象存储设备(OSD)。高效简洁的对象分布算法可以均衡 I/O 负载、自适应系统的变化、提高数据可靠性[3]。

对象分布算法是对象存储系统的核心组件，针对对象分布问题已有很多研究。为了解决对象分布的均匀性问题，提出一致性哈希算法[4]、层次化一致性哈希算法[5]、双模对象分布算法[6]、Crush 算法[7]、分层对象分布算法[8]、随机切片算法[9]等。为了提高存储 I/O 性能和降低数据热点问题，Xu 等提出了 EDP 算法以保证去重存储系统中均匀读[10]；Jouini 等提出了适用于文档存储的副本放置算法[11]；Gao 等提出了根据数据的访问频率分布对象算法[12]；Shen 等提出了一种应用于 BCube 网络拓扑结构的提高数据修复速率的对象分布算法[13]；Qin 等提出了对象分布算法分别针对纠删码存储中降级读和写性能进行优化[14]，以上提及的对象分布算法虽然各具特色，但是在对象存储系统的应用中受到诸多约束，不具有普适性。本文利用存储系统批量扩容和删除的特点，设计了弧映射双层对象分布(TMHR)算法。将同一批次的多个存储节点划分为子集群，对象分布过程中，TMHR 算法首先采用可扩展的子集群哈希(SHFC)算法选择存放对象的子集群，其次采用随机置换算法在目标子集群内计算存放对象的存储节点。该算法可以在较短时间内计算任意对象的目标 OSD 节点，保证对象均匀分布，迁移较少的对象以适应 OSD 节点的动态变化。

1　问题定义

设存储系统以子集群为单位进行扩容和删除，扩容、删除的第 i 个子集群用 C_i 表示，其权值用 w_i 表示，w_i 是子集群 C_i 内节点容量、性能等特征的量化，根据具体需求赋予权值不同的含义。如以存储容量为量化标准，则 1 TB 的存储节点和 500 GB 的存储节点的权值之比为 2∶1。C_i 拥有的存储节点数为 m_i。初始时存储系统仅包括子集群 C_0，所有子集群用子集群集合 C 表示，所有子集群数用 k 表示。对象数为 $y(y\geqslant 1)$，每个对象拥有唯一的识别符，对象数理论上无限，且对象数远远大于节点数。

2　TMHR 算法

利用存储系统批量扩容、删除的特性，TMHR 算法采用双层映射模式，存储节点按照加入系统的批次被分为多个子集群，相同的子集群内节点性能、容量相似。客户端读写对象时只需要从存储系统获取当前子集群和节点的描述信息，客户端在本地完成对象和其存储节点映射关系的计算，即在子集群间采用 SHFC 算法以权值为概率选取目标子集群，在目标子集群内采用随机置换算法等概率选择目标节点。

2.1　子集群间 SHFC 算法

基于动态区间映射算法和随机切片算法两者都将对象存储在以区间为单位的逻辑单元中，两种算法的时间复杂度与区间数均正相关。系统扩容过程中两种算法都会产生大量小区间，虽然随机切片算法提出了 4 种减小区间碎片化的策略，但是区间数仍然较大，极大地影响了系统 I/O 性能。为了解决系统扩容造成的区间碎片化问题，SHFC 算法将对象和子集群的哈希值映射在半径为 R 的环形空间，根据子集群的权值将整个环形空间分为相应比例的弧，每个子集群负责存储映射在弧上的对象，并将最大相邻弧合并策略应用于系统的扩容、删除过程。

初始时，SHFC 算法构造半径为 R 的环形空间，其弧长范围为 $[0,2\pi R]$，存储系统只有子集群 C_0，C_0 负责存储映射在整个环形空间的对象。当系统拥有子集群集合 $C=\{C_0,C_1,\cdots,C_{k-1}\}$ 时，加入权值为 w_k 的子集群 C_k 后，集合 C 内的子集群都要映射部分弧到子集群 C_k。子集群 $C_i(0\leqslant i\leqslant k-1)$ 需要映射的弧长为 $2\pi R(w_i/\sum_{j=0}^{k-1}w_j-w_i/\sum_{j=0}^{k}w_j)$，若子集群 C_i 的弧序列中可以组合出与需要映射的弧长度相等的弧集合，则直接将弧集合映射到子集群 C_k 上。若不存在，则采用最大相邻弧合并策略切割出符合要求的弧。

当系统内子集群集合 $C=\{C_0,C_1,\cdots,C_k\}$ 时，设删除子集群 C_j，则 C_j 的弧序列重映射到子集群 C_i $(0\leqslant i\leqslant k-1,i\neq j)$ 的弧长为 $2\pi R(w_i/(\sum_{t=0}^{j-1}w_t+$

$\sum_{t=j+1}^{k-1} w_t) - w_i / \sum_{t=0}^{k-1} w_t)$，若子集群 C_j 的弧序列可以组合出与映射到子集群 $C_i(0 \leqslant i \leqslant k-1, i \neq j)$ 的弧长度相等的弧集合，则直接将此集合映射到子集群 C_i $(0 \leqslant i \leqslant k-1, i \neq j)$ 上。若不存在，则采用最大相邻弧合并策略切割出符合要求的弧。

当映射在子集群 C_i 上的弧序列需要重映射部分弧到子集群 C_j 时，设需要迁移的弧长为 L，设子集群 C_i 上的弧序列为 $\{a_1, a_2, \cdots, a_e\}$，则最大相邻弧合并策略会尽可能的从子集群 C_i 的弧序列中找到多个弧 $a_u(1 \leqslant u \leqslant e)$ 之和的长度接近 L。当需要对存在的弧切割时，切割满足如下条件的弧，该弧切割后可以与目标子集群 C_j 的弧合并，同时该弧是所有满足条件的弧中最长的弧。

存储系统扩容、删除子集群后，SHFC 算法不需要重新分割整个环形区间，而是更新发生变化的弧和子集群之间的映射关系，这种策略使重映射的弧长度最短，从而使需要迁移的对象最少。

2.2 子集群内随机置换算法

子集群内的对象分布采用随机置换算法降低计算时间开销。随机置换算法实现了对副本机制的支持，保证同一对象的多个副本均匀分布在 OSD 节点上。计算对象 x 的第 r 个副本存放节点的函数为

$$f(x, r, m) = (\mathrm{hash}(x) + rp) \bmod m \tag{1}$$

式中：x 是对象的标识符；r 是小于 m 的副本数；p 和 m 都是质数，且满足 $p > m$。m 默认等于子集群 $C_i(0 \leqslant i \leqslant k-1)$ 的节点数 m_i，但是如果子集群 C_i 的节点数 m_i 并非质数，则 m 取大于 m_i 的最小质数。式(1)对序列 $\{0, 1, \cdots, m-1\}$ 进行随机置换生成等概率的新序列，对象 x 的第 $t(0 \leqslant t \leqslant r-1)$ 个副本存放在新序列的第 t 位。若新序列的第 t 位序号大于节点数 m_i，则顺次选择下一位。存储系统的扩容和删除等都是批量操作，很少或者几乎不会出现单个存储节点加入、删除的情况，单个存储节点故障失效后很快会被替换，子集群内的节点数是维持不变的，因此不需要考虑子集群内的节点变化情况。

3 算法理论分析

3.1 公平性和自适应性分析

设对象 x 第 r 个副本在子集群集合 C 中选择子集群 C_i，在 C_i 中等概率选择某一节点，则只需证明子集群 C_i 任意节点被选中的概率等于 $(1/m_i) \cdot (w_i / \sum_{j=0}^{k-1} w_j)$，即可证明 TMHR 算法满足公平性。

采用归纳法证明 SHFC 算法的公平性，因为对象哈希映射在环形空间上任意点的概率相等，所以对象映射到任一子集群的概率只和子集群的弧长有关。在第 2.1 节中，初始时系统只存在子集群 C_0，它负责存储所有对象，设存储系统经过 q 次扩容和删除操作后，系统内有 k 个子集群。子集群 C_i 被对象 x 选中的概率为 $w_i / \sum_{j=0}^{k-1} w_j$，只需证明第 $q+1$ 次系统变化后子集群 C_i 被对象 x 选中的概率仍然与权值成正比，可以分为 2 种情况考虑：①加入子集群 C_k 后，前 k 个子集群分别将自身弧序列中的一部分重映射到新加入的子集群，且第 i 个子集群 C_i 重映射的弧长为 $2\pi R(w_i / \sum_{j=0}^{k-1} w_j - w_i / \sum_{j=0}^{k} w_j)$，则新加入的子集群 C_k 拥有的弧长之和是前 k 个子集群重映射的弧长之和 $2\pi R(\sum_{i=0}^{k-1} (w_i / \sum_{j=0}^{k-1} w_j - w_i / \sum_{j=0}^{k} w_j)) = 2\pi R(w_k / \sum_{j=0}^{k} w_j)$；②子集群 C_j 被删除后，其弧序列分别映射到剩下的 $k-1$ 个子集群，子集群 C_i 拥有的弧长之和为 $2\pi R(w_i / \sum_{t=0}^{k-1} w_t + (w_j / \sum_{t=0}^{k-1} w_t) \cdot w_i / (\sum_{t=0}^{j-1} w_t + \sum_{t=j+1}^{k-1} w_t)) = 2\pi R w_i / (\sum_{t=0}^{j-1} w_t + \sum_{t=j+1}^{k-1} w_t)$，则第 $q+1$ 次加入或删除子集群后映射在子集群 C_i 的弧长仍然和其权值成正比。因此，可以证明 SHFC 算法满足公平性，并且根据上文可以看出，子集群发生变化后都是根据权值进行弧的重映射，对象迁移仅仅在发生变化的子集群和已有的子集群之间，未发生变化的子集群之间并未发生对象迁移，因此可以证明 SHFC 算法也是满足自适应性的。

以下证明随机置换算法满足公平性。第 2.2 节中随机置换算法的核心思想可以用式(1)表示，式(1)可以分解为如下 3 个运算操作

$$d = \mathrm{hash}(x) \bmod m \tag{2}$$

$$h = rp \bmod m \tag{3}$$

$$l = (d + h) \bmod m \tag{4}$$

式(2)表示以对象 x 为种子生成随机数，取模映射在 $\{0, 1, \cdots, m-1\}$ 整数范围内，其值为 d。式(3)表示以对象的副本为特征值对序列 $\{0, 1, \cdots, m-1\}$ 随机置换，文献[15]指出，如果 $\gcd(b, c) = 1$，$bp \equiv g \bmod c$ 并且 p 是解，那么 p 一定是唯一解，因此可以得出对象 x 的副本数 r 和 h 存在一一映射关系。式(4)表示将置换后的整数序列循环右移了 d 位得到 l。综上可得，式(1)以对象 x 和副本数 r 为特征

值，将序列$\{0,1,\cdots,m-1\}$等概率随机置换生成新序列。若 m 与子集群 C_i 的节点数 m_i 相等，则任意节点被选中的概率是 $1/m_i$，若 m 是大于 m_i 的最小质数，重复依次选择下一位，则任意节点被选中的概率是$(1/m)\cdot(m/m_i)=1/m_i$，因此随机置换算法满足公平性。

综上所述，对象 x 第 r 个副本分布时，子集群 C_i 被选中的概率为 $w_i/\sum_{j=0}^{k-1}w_j$，子集群 C_i 内任意节点被选中的概率为 $1/m_i$，则子集群 C_i 内任意一个节点被选中的概率等于 $(1/m_i)\cdot(w_i/\sum_{j=0}^{k-1}w_j)$，因此 TMHR 算法满足公平性。当系统扩容、删除时，对象的迁移也仅仅发生在变化的节点与未变化的节点之间，未变化的节点间没有对象迁移，因此 TMHR 算法也满足自适应性。

3.2　复杂度分析

TMHR 算法的对象分布时间由对象哈希映射到环形空间上的时间、SHFC 算法的计算时间和随机置换算法的计算时间 3 部分组成，对象哈希的时间可以忽略不计，SHFC 算法时间复杂度与环形空间弧数成正相关。随机置换算法只包含相加取模运算，时间复杂度为 $O(1)$。设某一时刻系统具有 $k-1$ 个子集群，子集群删除过程导致的弧数变化波动较小，因此我们只考虑系统不断扩容加入子集群的情况。根据第 3.1 节的内容，加入 C_k 子集群后，前 $k-1$ 个子集群都会将自身的弧序列部分重映射到 C_k 上。设子集群 $C_i(0\leqslant i\leqslant k-1)$的弧序列数为 P_i，因为子集群的权值不同，子集群可能需要切割自身的弧序列数变化也很大，可能重映射一条弧即可满足条件，也可能需要将大部分弧重映射才可以满足条件。即第 i 个子集群重映射的弧序列数 t 满足 $1\leqslant t\leqslant P_i$，则第 k 个子集群 C_k 的弧序列数 P_k 满足 $k\leqslant P_k\leqslant\sum_{i=0}^{k-1}P_i$，在不考虑弧合并的情况下，环形空间弧数 P 满足 $k(k+1)/2\leqslant P\leqslant\sum_{j=0}^{k}\sum_{i=0}^{j-1}(i(i+1)/2)$。弧的查找过程可以采用二叉树查找完成，因此可知 SHFC 映射算法时间复杂度为 $O(\mathrm{lb}k)$，则 TMHR 算法时间复杂度为 $O(\mathrm{lb}k)+O(1)=O(\mathrm{lb}k)$。

TMHR 算法需要保存 OSD 节点信息和环形空间弧信息，上文求得的环形空间弧数 P 满足 $k(k+1)/2\leqslant P\leqslant\sum_{j=0}^{k}\sum_{i=0}^{j-1}(i(i+1)/2)$，OSD 节点数为 $\sum_{i=0}^{k}m_i$，则 TMHR 算法的空间复杂度为 $O(k^2+\sum_{i=0}^{k}m_i)$。因为子集群数 k 远远少于节点数 $\sum_{i=0}^{k}m_i$，TMHR 算法的内存占用率主要和 OSD 节点数成正比，实际情况中本文算法的内存占用率也是相对较低的。综上可知 TMHR 算法满足高效性。

4　实验对比分析

在计算时间、均匀性、公平性和自适应性 4 个方面比较 TMHR 算法、一致性哈希算法和随机切片算法，实验的操作系统为 ubuntu 14.04 LTS 系统，采用 python 语言实现 3 种算法。

图 1 给出了由 120～360 个节点组成的存储系统中一致性哈希算法、TMHR 算法和随机切片算法分别分布 100 万个对象的计算时间比较。从图 1 可以看出，TMHR 算法是最快的，一致性哈希算法次之，随机切片算法的计算时间最长。主要是因为一致性哈希算法时间复杂度为 $O(\mathrm{lb}n)$，其中 n 为系统的 OSD 节点数，TMHR 算法的时间复杂度为 $O(\mathrm{lb}k)$，子集群数 k 远小于节点数 $n=\sum_{i=0}^{k}m_i$，而随机切片算法时间复杂度是 $O(\mathrm{lb}s)$，s 为区间数，区间数 s 和节点数 n 呈平方关系，实验结果和理论分析是相契合的。

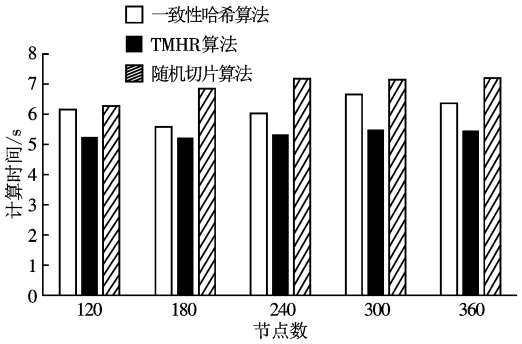

图 1　对象分布时间开销的比较

图 2 给出了将 15 万个对象分布到由 25 个节点组成的存储系统后的对象分布情况，25 个节点共分为 5 个子集群，每个子集群包括 5 个节点，子集群与子集群之间的 OSD 节点是异构的，5 个子集群的权值比为 1∶2∶3∶4∶5，从图 2 可以看出，对象在 0～4 号子集群内是均匀分布的，子集群间呈现比例分布，也就是说异构环境中对象的分布会随加入 OSD 节点特性的变化而变化，因此本文算法是完全适应于异构环境的。

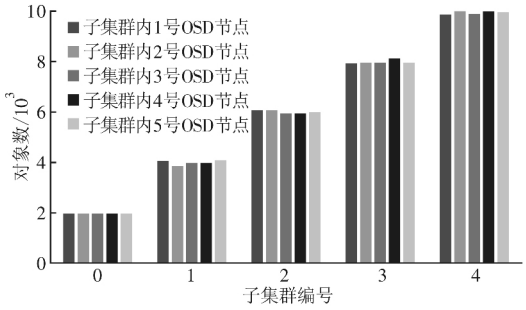

图2 不同权值OSD组成的子集群间对象数比较

图3给出了由120～360个节点组成的存储系统中3种算法分别分布100万个对象的均匀性比较。从图3可以看到，TMHR算法和随机切片算法的方差比一致性哈希算法的方差小，即这两种算法对象分布的更加均匀，有效保证了存储节点的负载均衡。TMHR算法和随机切片算法都可以通过理论证明节点分配的对象数和其权值成正比，并且TMHR算法将对象进行双层映射细粒度分布，比随机切片算法分布更加均匀。一致性哈希算法只是将节点、对象随机分布到环形空间，当数据量较大时对象才能均匀分布，因此其公平性相对较差。

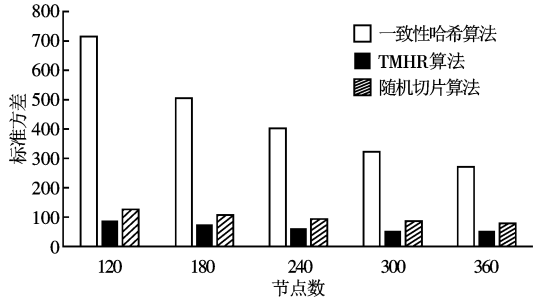

图3 对象分布算法的公平性比较

实验模拟比较一致性哈希算法、TMHR算法在存储系统批量扩容、删除后对象的迁移量，对象迁移量越少表示算法自适应性越好。图4、5给出了180个节点组成的存储系统扩容、删除后每个节点的对象迁移量。存储系统中每5个节点为1个子集群。图4是模拟存储系统的61～120号节点失效后剩余节点的对象迁移量。存储系统中预先已分布了100万个对象，因此理论上可以计算得到在节点删除前每个节点存储的对象数为$10^6/180\approx5\ 555$个。在节点删除后，每个节点存储的对象数为$10^6/120\approx8\ 333$个，则被删除的节点平均迁移2 778个对象到剩余的节点。从图4可以看出，TMHR算法在1～60号节点和121～180号节点的变化量在2 800左右，实际迁移量和理论值相近，而一致性哈希算法在1～60号节点和121～180号节点的变化量在5 000～6 700左右波动，迁移量较大。此外，61～120号节点的迁移量表示在节点删除前节点的对象数，可以看出在删除前TMHR算法的对象分布相对均匀，每个节点都大致分布了5 500个对象，和理论值5 555大致相等，而一致性哈希算法每个节点分布的对象数波动较大，公平性较差。

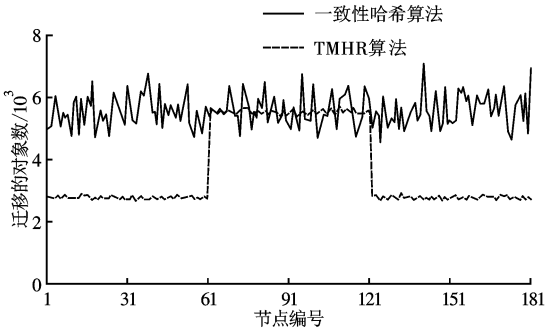

图4 61～120节点失效后的数据迁移量比较

如图5所示是模拟180个节点组成的存储系统加入60个节点后1～180号节点的对象迁移量。系统扩容前每个节点分配的对象数为$10^6/180\approx5\ 555$个。在节点增加后，每个节点存储的对象数为$10^6/240\approx4\ 166$个，因此扩容后每个节点都要迁移$5\ 555-4\ 166=1\ 389$个对象到新加入节点。从图5可以看出，TMHR算法中每个节点的对象迁移量和理论值相近，且每个节点迁移量也大致相等，而一致性哈希算法对象迁移量波动较大。

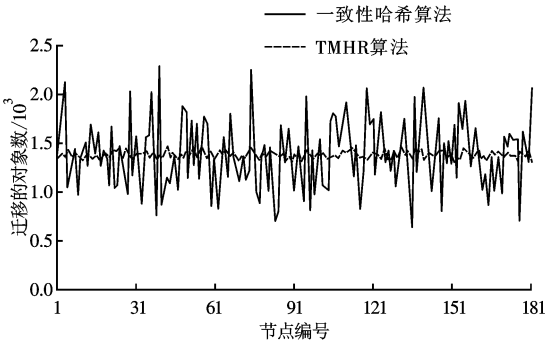

图5 节点增加后的数据迁移量比较

通过模拟实验，从图1～5可以得出如下结论：与一致性哈希算法和随机切片算法相比，TMHR算法在对象分布时间上分别缩短了20%和28%，较快的对象分布有利于客户端快速读写对象，提高系统性能；实验结果表明，TMHR算法可以使对象分布的更加均匀，保证系统负载均衡；当系统扩容、删除后，可以迁移较少的对象以保证对象均匀分布。

5 结　论

大数据时代下，对象分布算法对于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无法在公平性、自适应性、高效性和简洁性取得一定的平衡，很难应用于 EB 级混合异构存储系统中。本文提出了一种高效简洁的 TMHR 对象分布算法，算法支持权值和副本机制，在具有较低的时间复杂度的前提下，兼顾了算法的公平性、高效性、简洁性和自适应性。实验模拟结果表明，与一致性哈希算法和随机切片算法相比，TMHR 算法在对象分布时间上分别缩短了 20%和 28%，数据的分布也更加接近于理论情况，对象迁移量方面更加接近理论最优值。异构存储系统中对象分布数量与 OSD 节点的权值成正比，因此 TMHR 算法更加适用于异构对象存储系统。

参考文献：

[1] WEIL S A, BRANDT S A, MILLER E L, et al. Ceph: a scalable, high-performance distributed file system [C]// Proceedings of the 7th Symposium on Operating Systems Design and Implementation. New York, USA: ACM, 2006: 307-320.

[2] LAKSHMAN A, MALIK P. Cassandra: a decentralized structured storage system [J]. ACM SIGOPS Operating Systems Review, 2010, 44(2): 35-40.

[3] FACTOR M, METH K, NAOR D, et al. Object storage: the future building block for storage systems [C]// Proceedings of the 2005 IEEE International Symposium on Mass Storage Systems and Technology. Piscataway, NJ, USA: IEEE, 2005: 119-123.

[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]// Proceedings of the 29th ACM Symposium on Theory of Computing. New York, USA: ACM, 1997: 654-663.

[5] ZHOU J, XIE W, GU Q, et al. Hierarchical consistent hashing for heterogeneous object-based storage [C]// Proceedings of the 14th IEEE International Symposium on Parallel and Distributed Processing with Applications. Piscataway, NJ, USA: IEEE, 2016: 1597-1604.

[6] XIE W, ZHOU J, NOBLE J, et al. Two-mode data distribution scheme for heterogeneous storage in data centers [C]// Proceedings of the 2015 IEEE International Conference on Big Data. Piscataway, NJ, USA: IEEE, 2015: 327-332.

[7] WEIL S A, BRANDT S A, MILLER E L, et al. CRUSH: controlled, scalable, decentralized placement of replicated data [C]// Proceedings of the 2006 ACM/IEEE Conference on Super-Computing. Piscataway, NJ, USA: IEEE, 2006: 31.

[8] **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 [J]. 计算机研究与发展**, 2012, 49(4): 887-899.
CHEN Tao, XIAO Nong, LIU Fang. An efficient hierarchical object placement algorithm for object storage systems [J]. Journal of Computer Research and Development, 2012, 49(4): 887-899.

[9] MIRANDA A, EFFERT S, KANG Y, et al. Random slicing: efficient and scalable data placement for large-scale storage systems [J]. ACM Transactions on Storage, 2014, 10(3): 1-35.

[10] XU M, ZHU Y, LEE P P C, et al. Even data placement for load balance in reliable distributed deduplication storage systems [C]// Proceedings of the 23th IEEE International Symposium on Quality of Service. Piscataway, NJ, USA: IEEE, 2016: 349-358.

[11] JOUINI K. Distorted replicas: intelligent replication schemes to boost I/O throughput in document-stores [C]// Proceedings of the 2017 IEEE/ACS International Conference on Computer Systems and Applications. Piscataway, NJ, USA: IEEE, 2017: 25-32.

[12] GAO Y, LI K, JIN Y. Compact, popularity-aware and adaptive hybrid data placement schemes for heterogeneous cloud storage [J]. IEEE Access, 2017, 5(99): 1306-1318.

[13] SHEN Z, LEE P P C, SHU J, et al. Encoding-aware data placement for efficient degraded reads in XOR-coded storage systems [C]// Proceedings of the 35th IEEE Symposium on Reliable Distributed Systems. Piscataway, NJ, USA: IEEE, 2016: 239-248.

[14] QIN Y, AI X, CHEN L, et al. Data placement strategy in data center distributed storage systems [C]// Proceedings of the 2016 IEEE International Conference on Communication Systems. Piscataway, NJ, USA: IEEE, 2017: 1-6.

[15] STINSON D R. Combinatorial designs: constructions and analysis [M]. Berlin, Germany: Springer-Verlag, 2003: 56-57.

（编辑　武红江）