文章编号:2095-6134(2016)05-0686-07

一种均衡视频资源的分布存储方法*

侯金钟†, 张立波, 罗铁坚

(中国科学院大学计算机与控制学院, 北京 101408)

(2016 年 1 月 25 日收稿; 2016 年 3 月 18 日收修改稿)

Hou J Z, Zhang L B, Luo T J. A distributed storage method of balancing video resources[J]. Journal of University of Chinese Academy of Sciences, 2016,33(4):686-692.

摘 要 随着互联网视频内容日益增多,视频资源的分布式存储受到关注.在分布式存储系统中,如果文件能均衡存储在各个节点会使系统更加健壮,而传统的分布式存储系统通常在失衡发生后再调整,带来了较多的 IO 开销.对分布式视频文件存储进行研究,提出一种利用哈希和 Bloom Filter 的高性能存储系统 HBF,在文件存入系统时即进行存储平衡.系统具备多个节点,文件分散保存在不同的节点上,系统通过增加或删除节点使容量具备可伸缩性,而且在存储平衡方面进行了改进,使存储节点之间的存储使用量保持相对一致.实验证明,HBF 使分布式视频文件存储系统具有高性能并兼顾节点存储平衡,有利于负载均衡和资源的合理利用.

关键词 视频资源; 分布式存储; 存储均衡; 高性能

中图分类号:TP393 **文献标志码**:A **doi**:10.7523/j.issn.2095-6134.2016.05.017

A distributed storage method of balancing video resources

HOU Jinzhong, ZHANG Libo, LUO Tiejian

(*School of Computer and Control Engineering, University of Chinese Academy of Sciences, Beijing* 101408, *China*)

Abstract With the increase of internet video contents, the method of distributively storing these videos has attracted much attention. In the distributed storage system, storage balance will make a system more robust. However traditional distributed storage systems always adjust the storage imbalance after it happens, which might cause more IO cost. This study focuses on distributed video file storage and proposes a kind of storage system based on Hash and Bloom Filter, namely HBF. It balances the storage when a file is being stored into the system. The system contains different nodes and the files are distributively stored on these nodes. The ability to easily add and remove nodes makes capacity of the system more scalable. In addition, improvments in storage balance have also been achieved, which keeps the usage of nodes storage relatively consistent. The experiments indicate that HBF achieves a good balance of nodes storage as well as the high performance of the distributed video files storage system. Thus the load balance could be greatly improved, and meanwhile the ultilization of resources would be more reasonable.

* 中国科学院设备共享管理系统优化项目(Y42901VED2)资助

† 通信作者,E-mail:houjinzhong13@mails.ucas.accn

Key words　video resources; distributed storage; storage balance; high performance

视频作为一种广受欢迎的资源在网络上日益流行,电影、电视剧、动漫、娱乐节目、纪录片等都以视频的形式呈现在我们面前. 这些视频带给我们视觉和听觉的享受,丰富了现代生活.

另一方面,视频在提供丰富生动的内容的同时也带来了对其进行存储和访问的挑战. 由于视频文件同时包含图像和声音,存储时需要更多的存储空间. 尽管大多数视频均使用压缩格式进行存储,但其对空间的要求依然远高于一般文件. 同时,对于视频提供商来说,用户观看视频对网络状况有一定的需求. 如果网络状况较差,视频流接收速率低,容易导致播放卡顿,严重影响用户体验.

为解决视频存储空间和传输成本的问题,DAN(Directed Access Network)被提出,DAN 使用分布式的可扩展的方式存储视频文件以解决存储空间的问题,并设置一系列的边缘节点作为视频文件缓存,使得用户的请求可以就近得到响应,提升用户的播放体验[1]. 在先前版本的 DAN 系统[2]中,视频文件的存储和检索是非常重要的一个方面,尽管使用 Bloom Filter 进行优化,仍然需要对存储节点进行由近及远的逐个查找,效率较低;同时在给视频文件选择存储节点的过程中仅考虑了地域分布,未考虑到存储节点的存储均衡.

针对以上问题,本文提出一种基于哈希和 Bloom Filter 的文件系统 HBF(Hash and Bloom Filter Based File System). HBF 可以应用于原有 DAN 中取代原有的文件系统,它同 DAN 中的文件存储系统一样具备可伸缩易扩展的特性,同时改进了其文件存储和检索的性能,文件在多个节点进行存储时更加均衡. HBF 取代 DAN 系统中的存储节点,并结合原有的总控节点和边缘节点,可以构建一个更加高效稳定的流媒体分发系统. 其中总控节点负责维护全局元数据和视频列表,边缘节点提供缓存和用户接口,存储节点高效均衡地进行文件存储,所有这些节点共同协作,为用户提供流畅的视频播放体验.

本文主要就分布式视频存储的策略进行研究,探索 HBF 的节点设置、存储方法、可伸缩性和检索方法等问题.

1　相关原理

1.1　Bloom Filter

Bloom Filter 是一种能够快速查询某元素是否包含于特定集合的数据结构[3-4]. 它使用一组哈希函数和一个向量来表示集合数据,其中的向量为二进制向量,即某个维度的值为 0 或 1.

在将元素 key 插入集合的时候,首先通过 k 个哈希函数分别计算得到 k 个哈希值,然后将 k 个哈希值作为下标对应至二进制向量的 k 个位置并将这些位置的值置为 1.

在查询某个元素 key 是否存在于集合中的时候,同样通过 k 个哈希函数得到 k 个值,并检查以这些值为下标的位置上的值是否均为 1,如果是则返回元素存在,反之如果存在不为 1 的位置,则要查询的元素不在集合内.

但是 Bloom Filter 有可能出现伪正例,即在元素不存在集合中时有很小的可能返回元素存在. 针对这一问题,可以通过优化设置 Bloom Filter 的各项参数改善. 假定 Bloom Filter 结构含有 m 个二进制位,k 个哈希函数,且有 n 个元素需要存入,则最优哈希函数个数为

$$k = \frac{m}{n}\ln 2,$$

给定伪正例出现概率 p,最优的二进制位个数为

$$m = -\frac{n\ln p}{\ln 2}.$$

此外,如果给定了 m, n, k,有

$$p = (1 - e^{-k(n+0.5)/(m-1)})^k.$$

1.2　一致性哈希

一致性哈希是应用在分布式系统中的算法[5-6]. 它首先随机或按一定策略产生一组 token,然后为每个节点分配一个 token. 在进行文件存储的时候,通过哈希函数将文件映射到哈希空间的某个值,并查找到第一个不小于该值的 token,该 token 所对应的节点即为文件将要存放的节点. 文件检索的流程与存储的过程类似,通过计算要查找的文件的哈希值,找到文件可能存放的位置,如果该位置存在要查找的文件则查找成功,否则系统中未存有该文件.

在实际应用中,为了错误恢复一般要进行文

件的冗余存储. 对于一个文件,一般采用 3 个副本的策略进行存储. 一致性哈希中的通常做法是在文件存储到某个节点时,将这一节点之后的 2 个节点同时存储一份拷贝.

一致性哈希中相对容易出现各节点存储不平衡的问题,解决办法通常是将存储压力过大的节点的哈希空间分裂到其他节点,与此同时,对应的文件也需要进行物理转移,这种方式增加了维护存储均衡的开销.

1.3 Kinesis

Kinesis 是微软研究院提出的分布式存储系统[7]. 在 Kinesis 中,服务器节点被划分为 k 个组,每组的存储量基本一致. 每个组具有一个与其他组不同的线性哈希函数,能够将文件映射到组内某个节点. 在进行文件存储的时候,对于某一特定文件,假设其备份策略为 r 备份,则在划分好的 k 个服务器组中计算出 k 个位置,每个位置对应特定服务器组内的某个节点,通过计算这些节点的负载,将 r 个负载小的节点选取出来,并将文件存储在这 r 个节点中. 通过这种存储策略,使得文件在存储的时候即考虑到负载均衡,避免了在负载不均衡发生后进行文件调度以重新获得均衡所需的开销.

Kinesis 在文件进行存储时即充分考虑负载均衡,避免了后期因负载不均衡而造成的文件移动开销. 但是在该系统中增加节点不够灵活,往往需要增加原有节点一倍数量的节点,当特定哈希空间不均衡时不能通过简单地增加一个或少量几个节点来解决.

2 HBF 存储原型系统

对于一个分布式文件系统来说,在设计实现的时候需要考虑的因素有很多,重要的如可扩展性、负载均衡、存储和检索效能、错误恢复等,在工业环境中还需考虑更多的因素如配置管理、系统监控与报警、任务调度等[8-9]. 本文主要就 HBF 的文件操作接口、扩展性、存储划分策略、查询策略等方面做重点研究.

2.1 系统接口

HBF 由一系列的平行节点组成,各个节点的角色分工相同,提供文件的存取功能. 它是一个基于 key-value 的存储系统,value 对应具体的文件,key 是文件相关联的识别标志(如文件名,MD5 值等). 和大多数 key-value 的存储系统一样,HBF 的系统接口十分简洁,它只有 2 个接口方法:一个是 put,一个是 get. Put 方法将一个文件放入文件系统,get 方法将一个 key 对应的文件取出.

2.2 存储策略

HBF 在设计时充分考虑了文件在各个节点之间的平衡分布,在一致性哈希的基础上加以创造性改进,使得存储负载更加均衡.

在一致性哈希算法中,哈希函数的取值空间可以看做是一个环形. 系统从这个取值空间中随机产生 k 个 token,每个 token 分配给一个服务器节点. 当调用 put 操作存放文件时,由文件名或文件的其他特征得出文件的 key,再由 hash 函数计算出其对应的 value 值. 然后按照从小到大的顺序查找 token,找到第一个不小于 value 的 token,在这个 token 所对应的节点存放文件,形成 key 到 value 再到节点的对应关系.

在传统的系统中进行副本存储时,假设副本个数为 M,经典的做法是先按照一致性哈希算法计算得到一个节点,然后在哈希环上按照顺时针方向依次找到该节点的 $M-1$ 个后继节点,并在这些后继节点上存储文件的副本[10]. 按照这种方式进行存储划分容易出现的问题是文件存储不均衡,为此很多使用一致性哈希的系统设置了虚拟节点[11],并在不均衡产生时调整虚拟节点与实际物理节点的对应关系,并将对应的文件在物理机器间加以传递,使得存储负载大的节点上的文件分摊到负载小的节点上. 这种事后补救的做法一定程度加重了节点间的 IO 负担.

在 HBF 中存储副本的做法与此不同,如图 1 所示,HBF 引入了候选节点的概念,文件在存储时会从多个候选节点中选择其中部分节点存储副本. 假设副本数量为 M,候选节点数为 $K(K>M)$,它从 K 个候选节点中选择存储负载小的 M 个节点,并将副本存储于这些节点,给存储平衡以新的机会. 在 HBF 中,副本个数 M 和候选节点个数 K 可以由用户自定义. 在将文件存储至某一节点时,该节点及其顺时方向上的 $K-1$ 个节点组成候选节点,这些节点可能有不同的可用容量,从中选择存储负载小的 M 个进行实际的存储可以使得这些节点间保持相对的存储平衡.

通过改变 K 和 M 的值,可以得到不同的系统特性,获得不同的效果:

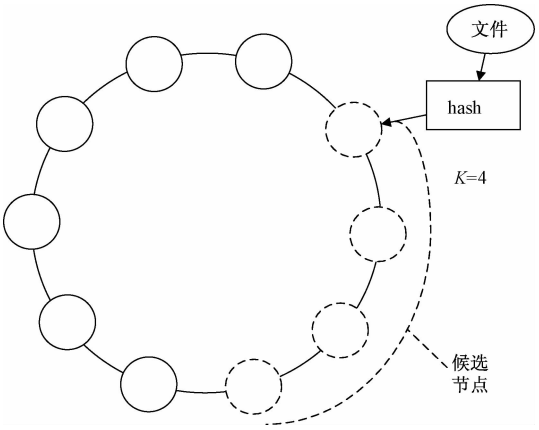

图 1　HBF 存储策略图示

Fig. 1　HBF storage strategy

1)如果 K 等于 M,系统等价于采用传统的一致性哈希算法;

2)如果 $K > M$,可以使得文件在存储时充分考虑节点间的存储平衡;

3)若 N 等于系统的节点总数,相当于在整个系统中选择存储压力最小的节点进行存储,系统将达到相当理想的存储平衡程度.

值得注意的是,在取得存储平衡的同时,由于增加了从 K 个候选节点中选取 M 个副本节点的过程,效率会有轻微的损失.但是由于增加的计算量为常数量级,且常数项很小,一般情况下对性能的影响可以忽略不计.

2.3　增加节点

在 HBF 中增加节点时,会优先考虑从负载较大的节点处划出一部分哈希空间给新增加的节点,以分担其负载,划分的比例可简单地为原节点哈希空间的一半,或根据原节点负载压力的紧迫程度和新节点的能力适当调整,并将原节点上的文件根据哈希值重新分配,计算得到的哈希值落在新节点的转存在新节点上.由于新增了节点,会导致原有的存储结构部分失效.

具体地,假设新加入的节点编号为 i,那么对于节点 i 逆时针方向数第 K 个节点,其候选节点由于新节点的加入变为了 $(i-K)\sim(i+1)$,为了维持原有的规则不变,将第 $i+1$ 个节点中作为第 $i-K$ 个节点副本的文件删除,并在第 $i-K$ 到 i 个节点中为其增加一个副本,这里选取节点的方式与 2.2 节中介绍的相同.

2.4　移除节点

节点需要移除的情况有两种,一种是由于系统缩小规模降低成本等原因永久性地移除节点,另外一种是由于节点故障导致节点临时不可用,但在较短时间内能够重新上线.

对于第 1 种情况,可将要移除的节点的哈希空间划分到相邻节点,并将对应的文件转移,如果文件从 A 节点移入 B 节点,但 B 节点已经含有该文件,那么就按照 2.2 中的方法再次选取一个节点,来维护副本数量不变.对于第 2 种情况,如果能够在短时间内将故障节点恢复,则可不对故障节点内的文件进行转移,仅将故障节点标记为不可用节点,不作为文件存取的候选节点.

2.5　视频文件检索

对于用户请求的视频,系统必须能够找到其存放的位置,才能为用户提供相应的视频流服务.对于某个视频,其存储位置可由哈希值得到.具体的做法为:

Step1　采用与视频文件存储时相同的哈希函数,对用户请求的视频产生一个哈希值 value;

Step2　按照从小到大的顺序查找存储节点,找到第一个 token 值不小于 value 的节点,该节点及其后续的 $N-1$ 个节点即为候选节点;

Step3　从 N 个候选节点中找出含有用户请求的视频的 M 个节点.

在 Step3 中,由于每个节点都对应有 Bloom Filter 结构,可以通过查找候选节点的 Bloom Filter 数据,判断其是否含有用户要找的视频[12].

在 DAN 系统中,需要逐个节点通过 Bloom Filter 进行查找,而 HBF 则通过采用一致性哈希快速定位候选节点,避免了在 DAN 中低效的检索过程,使得系统效能得以提升.

3　实验

为了验证 HBF 在文件存储中所能达到的效果,我们设计了 2 个实验.第 1 个实验对 HBF 算法的效率进行验证分析,第 2 个实验测试 HBF 在不同条件下所能达到的平衡性.

3.1　实验环境

本文主要针对检索效率和存储平衡性进行算法的改进,为了更好地验证算法在多种不同环境下的表现,也由于硬件数量的限制,所以本实验采用仿真验证的方式进行.具体实验软硬件环境如表 1 所示.

表 1 实验环境表

Table 1 Experimental environment

项目	参数
CPU	8 核
内存	8 GB
操作系统	Windows 7
程序设计语言	Java

3.2 实验数据

为使数据更加贴近真实应用场景,通过网络爬虫抓取国内视频网站爱奇艺的部分视频数据,视频量共计 34 124 个. 在这些视频中,最长视频约 5.4 h,最短视频仅 2 s. 具体的时长分布如图 2 所示,其中 1 ~ 5 min 之间的短视频数量最多,占视频总数的 54.8%.

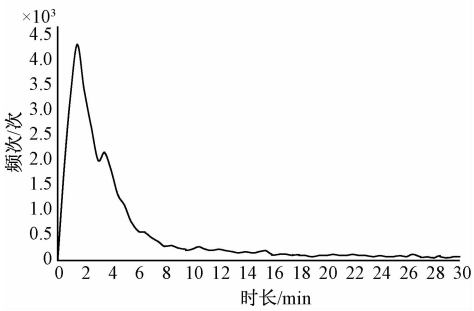

图 2 实验数据的分布状况

Fig. 2 Distribution of experimental data

这种数据的分布规律符合互联网视频的特点,契合 HBF 系统针对互联网视频资源进行存储的使用场景. 也正是由于这种数据特点,HBF 不进行文件的分块存储. 对文件先分片再进行存储在分布式系统中较常见,但这类系统一般适用于存储大型文件,而 HBF 针对的是众多短小的互联网视频,这是 HBF 不进行分片的主要考量因素. 同时,小文件多大文件少,也使系统在偶尔遇到大文件时可能会产生平衡波动,这种现象将在后续的实验中详细阐述.

3.3 检索性能实验

在文件检索性能测试中,模拟用户访问文件的动作,重复进行 30 万次查询测试,并记录系统在查询过程所花费的平均时间. 同时使用 DAN 和一致性哈希算法进行同样的实验. 表 2 是当节点数量为 200 个时各个不同策略的查询时间对比.

通过表格数据可以发现 HBF 在 3 种方法中显著优于 DAN 原有存储方案,与一致性哈希相比仅有微小差距.

表 2 检索时间表

Table 2 Query cost time ms/万次检索

策略	查询时间
HBF	448.716 4
DAN	717.442 3
一致性哈希	444.818 9

为验证节点数量对不同方法检索的影响,分别在节点数为 100,150,300,350,400,450,500 时对查询所需时间进行测量,得到图 3 所示的结果. 从图 3 可以发现,HBF 具有接近常数级的查找时间.

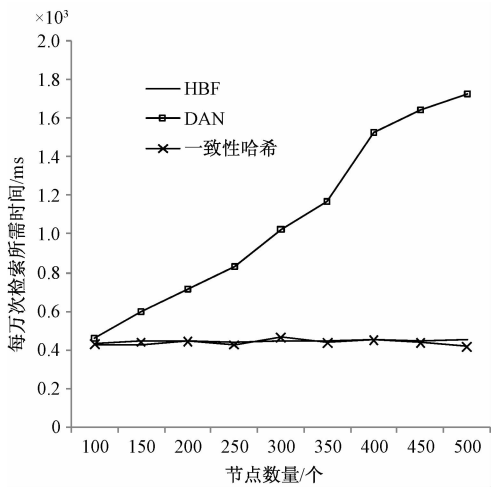

图 3 检索时间随节点数量的变化情况

Fig. 3 Variation in guery time with node amount

DAN 原有查找算法的查找时间会随着节点的增加而增加,而 HBF 和一致性哈希在算法的运行时间方面不依赖于节点的数量,且二者具有几乎一致的时间效率.

HBF 在效率方面较 DAN 有量级的提升,与一致性哈希几乎相同,这一结果与算法本身的时间复杂度吻合. 因为 HBF 和一致性哈希都是使用哈希函数直接定位节点,其中 HBF 额外包含了 Bloom Filter 查找过程,而哈希函数和 Bloom Filter 的时间复杂度为 $O(1)$,因此整体复杂度仍然是 $O(1)$;DAN 原有存储系统需要逐个节点进行查找,这一过程时间复杂度为 $O(N)$,因此呈现出随着节点数量增加,文件查找所需时间线性增长的现象.

3.4 存储平衡实验

传统的一致性哈希中,存储平衡的维护要推迟到存储失衡时再进行节点间文件的调度,对 IO

资源的消耗大. 相对于一致性哈希,HBF 最大的优势是在文件存储进系统时即对存储平衡进行更严格的维护,节省后续调整所需的 IO 资源.

在平衡性实验中,分别控制候选节点数量,存储副本数量,并与 DAN 和一致性哈希进行对比,记录系统失衡节点数量随时间的变化. 衡量存储平衡的指标借鉴 Hadoop 采用的衡量标准,用每个节点的存储使用率(节点磁盘使用量/节点存储总量)与整个系统的存储使用率(系统已使用的存储空间/系统存储空间总量)进行比较,如果差异超过一定阈值,则认为节点存储失衡. 设节点存储使用率为 u,系统存储使用率为 v,阈值为 t 则

$$|u-v|=\begin{cases}\leqslant t, & \text{平衡}\\ >t, & \text{失衡}\end{cases}.$$

设定实验阈值 t 为 5%,按照以上衡量指标,首先在不同候选节点数量下进行实验. 图 4 是在节点数量为 100,实际副本数为 3,控制候选节点数 K 分别为 4,5,6,7 情况下的实验结果.

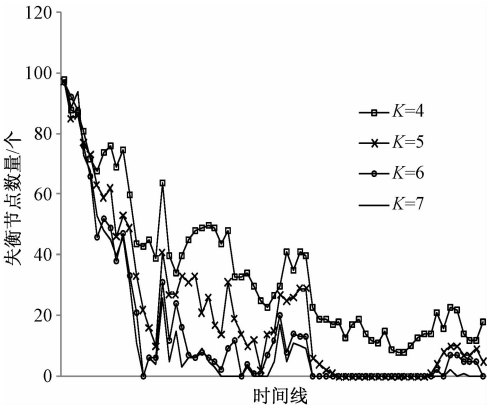

图 4　不同候选节点数量对应的存储平衡情况

Fig. 4　Storage balance for different candidate node numbers

通过该实验可以得出如下规律:

1)系统内失衡节点数量随候选节点数增加呈递减的趋势,候选节点数越多,系统的整体平衡性越好;

2)随着时间的推移,平衡性达到更好的状态,且更趋于稳定;

3)随着候选节点数增多,平衡效果加强但增量逐渐减少,在 $K=6$ 和 $K=7$ 时仅有较小差别. 在 K 取更大值的时候我们也进行了测试,发现平衡性没有显著提高.

通过上述实验可以发现 HBF 存在冷启动的问题,即在系统开始存放文件时,会存在较大的不平衡性,这是由于系统在没有存储任何文件时,每个节点的存储量均为 0,处于完全平衡状态,此时放入任何文件都很容易打破这个平衡. 随着时间的推移,系统内存储的文件较为均匀地分布于整个系统,使系统在平衡性方面更加健壮.

为保证系统的可靠性,存储在系统内的文件一般设置为 3 个副本,但有时因为特殊需求,副本的数量也会适当调整. 分别令副本数量为 2,3,4,5,并测试其对平衡性的影响,结果如图 5 所示.

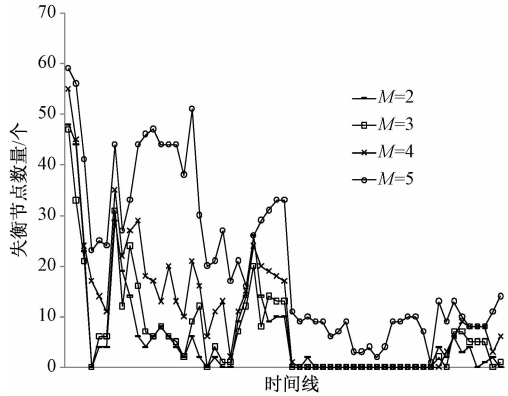

图 5　不同副本数对平衡性的影响

Fig. 5　Influence of copy number on balance

从图 5 可以发现,副本数量为 2、3、4 时存储平衡效果较好,而副本数量为 5 时存储失衡节点数增加,平衡效果相对较弱. 因为副本数为 5 时,6 个节点中有 5 个节点必须要存储文件,可供选择的余地小,因此平衡性差.

同时也可以注意到,在系统的运行过程中,失衡节点数量偶尔有所震荡,通过查询震荡点的数据发现通常是在震荡点存储了大文件,导致存储平衡受到影响.

为比较 HBF 的平衡策略与传统的一致性哈希以及 DAN 的平衡策略的差异,进一步利用上述实验数据,并设定 HBF 的副本数为 3,候选节点数为 6,做了 3 种方法的对比实验. 实验结果如图 6 所示. 图 6 表明,HBF 在存储平衡方面显著优于 DAN 和一致性哈希,采用 HBF 的存储平衡策略后将对集群节点资源的利用更加均衡合理,使得各个节点的能力得到更加充分的利用.

系统的失衡率为失衡节点数量与总节点数量的比值,这一指标可以用来有效衡量系统的存储平衡性. 通过对连续时间点的采样统计,在本实验条件下一致性哈希的节点失衡率为 64.53%,DAN 的失衡率为 60.31%,HBF 的失衡率为 5.95%.

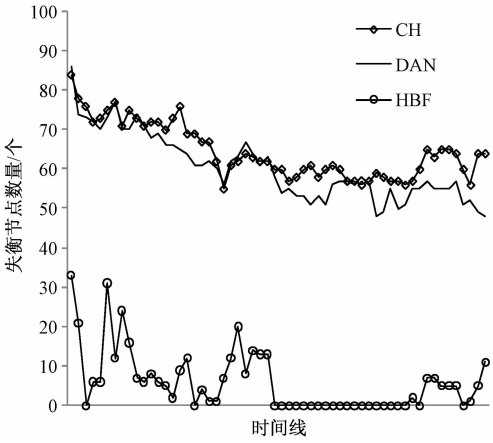

图 6 不同方法的存储平衡性

Fig. 6 Storage balance by using different methods

相对于一致性哈希和 DAN,HBF 在节点平衡性方面分别提高 10.8 倍和 10.1 倍.

4 总结和展望

HBF 提供了一个分布式环境下的视频文件存储策略,它为文件的存取提供了简洁的接口,文件能够以平衡、高效的方式存储在不同的节点上,存储节点能够灵活地增加或删除,使得系统具备可伸缩的容量,具有良好的可扩展性.

与一致性哈希相比,HBF 通过有针对性地选择存储节点来存放文件,使得存储平衡性得到显著的改善,相对于 Kinesis,它不需要每次增加一倍的节点来扩充容量,使得系统具有更好的可伸缩性,而且在性能方面,HBF 通过快速的哈希和 Bloom Filter 的结合,提供了与一致性哈希一致的性能.

HBF 具有均衡存储的效果,同时具备高效的性能.但在处理大文件时容易造成系统的震荡,今后将针对该问题进行重点研究.

参考文献

[1] Wang Z, Luo T. Intelligent video content routing in a direct access network[C]// Symposium on Web Society. 2011:147-152.

[2] Hou J, Luo T, Wang Z, et al. An intelligent media delivery prototype system with low response time[C]// Advances in Swarm and Computational Intelligence. Springer International Publishing, 2015:253-264.

[3] Bloom B H. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM, 2010, 13(7):422-426.

[4] Qiao Y, Li T, Chen S. Fast bloom filters and their generalization [J]. IEEE Transactions on Parallel & Distributed Systems, 2014, 25(1):93-103.

[5] 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 Twenty-Ninth Annual ACM Symposium on the Theory of Computing. 1997:654-663.

[6] Zhao N, Wan J, Wang J, et al. GreenCHT: a power-proportional replication scheme for consistent hashing based key value storage systems[C]// Mass Storage Systems and Technologies (MSST), 2015 31st Symposium on. 2015:1-6.

[7] Maccormick J, Murphy N, Ramasubramanian V, et al. Kinesis: a new approach to replica placement in distributed storage systems[J]. ACM Transactions on Storage (TOS), 2009, 4(4):1-28.

[8] Weil S A, Brandt S A, Miller E L, et al. CRUSH: controlled, scalable, decentralized placement of replicated data[C]// SC 2006 Conference, Proceedings of the ACM/IEEE. IEEE, 2006:31-43.

[9] Sun B J, Wu K J. Research on cloud computing application in the peer-to-peer based video-on-demand systems[C]// Intelligent Systems and Applications (ISA), 2011 3rd International Workshop on. IEEE, 2011:1-4.

[10] 王君君. 网络文件的分布式存储设计与实现[D]. 济南:山东大学,2015.

[11] Decandia G, Hastorun D, Jampani M, et al. Dynamo: amazon's highly available key-value store[J]. ACM Sigops Operating Systems Review, 2007, 41(6):205-220.

[12] Wang Z, Luo C, Luo T, et al. A bloom filter-based index for distributed storage systems[C]// Distributed Computing and Artificial Intelligence, 12th International Conference. Springer International Publishing, 2015:293-301.