一种混合局部恢复码及Hitchhiker码的存储策略

王梓仲[1),2)] 王海霞[2)] 邵艾然[1),2)] 汪东升[1),2)]

[1)](清华大学计算机科学与技术系 北京 100084)

[2)](清华大学北京信息科学与技术国家研究中心 北京 100084)

摘 要 我们正处于一个大数据的时代. 如今一个分布式存储系统需要存放PB数量级数据的情况越来越常见. 这些系统一般由普通商用组件构成,其出错率相对较高. 由此,分布式存储系统需要保证数据的可靠性和可用性. 多副本和纠删码是现在最为常用的技术. 相比多副本技术,采用纠删码能在同等容错能力下大幅降低存储开销. 然而,在进行数据恢复时,使用传统的纠删码(如Reed-Solomon码)会导致系统中产生大量的网络带宽消耗及磁盘读写操作,进而导致退化读延迟过高. 注意到在系统中数据的访问频率呈Zipf分布,大多数数据访问只涉及到少量数据,而绝大多数数据的被访频率很低. 根据这种数据访问的偏斜性,本文提出如下存储策略以解决采用纠删码的系统退化读延迟过高的问题:对被访频率高的热数据采用低恢复延迟的纠删码(如局部恢复码Local Reconstruction Code,LRC)进行编码,而对被访频率低的冷数据采用保证最小存储开销的纠删码(如Hitchhiker码)进行编码. 由于热数据占据了绝大多数的数据访问,因此绝大多数的退化读也将应用在这些热数据上,这样这一策略就能在整个系统的角度获取低恢复开销的优势. 同时,冷数据占据了系统绝大多数的数据量,且冷数据由保证最小存储开销的编码进行存储,因此这一策略的存储开销会很低. 然而,对于混合存储策略而言,热数据可能会变冷,而冷数据也可能会变热,因此它需要配置一种编码切换过程. 一个不恰当的编码切换过程会引起巨大的数据传输量,这是难以让人接受的. 为了避免这一缺陷,本文提出了一种LRC和Hitchhiker码之间的高效切换算法. 这一算法可以避免上述策略在部署时因冷热数据的转换出现系统瓶颈. 在精心选取了两种编码并提出它们之间的高效切换算法后,本文提出的混合存储策略避免了现阶段其余混合存储策略的主要缺点. 通过实验验证,此存储策略相较传统的Reed-Solomon码在退化读延迟方面降低了55.8%. 在编码切换方面,切换延迟能分别降低为重新编码算法用时的13.4%及33.1%,且当数据从LRC切换为Hitchhiker码时(更为频繁出现的情况)的数据传输量能降至10%.

关键词 纠删码;退化读;编码切换;容错;存储

中图法分类号 TP302.8 **DOI号** 10.11897/SP.J.1016.2020.00618

A Local-Reconstruction-Code-and-Hitchhiker-Code-Mixing Storage Scheme

WANG Zi-Zhong[1),2)] WANG Hai-Xia[2)] SHAO Ai-Ran[1),2)] WANG Dong-Sheng[1),2)]

[1)](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

[2)](*Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing* 100084)

Abstract We are now living in an era of big data. It is more and more common that distributed storage systems store multiple petabytes of data today. These systems are typically built on commodity components, where faults are more likely to occur. It is the responsibility of the distributed storage systems to ensure that data can be stored reliably and durably. Replication and erasure codes are the most common solutions. Under the same level of fault tolerance, using erasure

收稿日期:2018-10-26;在线出版日期:2020-02-07. 本课题得到国家重点研发计划项目(No. 2016YFB1000303)、广东省重点研发计划项目(No. 2018B010115002)资助. **王梓仲**,硕士研究生,计算机学会(CCF)会员,主要研究领域为存储系统容错. E-mail: wangzizh17@mails.tsinghua.edu.cn. **王海霞**,博士,副研究员,计算机学会(CCF)会员,主要研究领域为计算机体系结构. **邵艾然**,博士研究生,计算机学会(CCF)会员,主要研究领域为计算机体系结构. **汪东升**(通信作者),博士,教授,计算机学会(CCF)会员,主要研究领域为计算机体系结构. E-mail: wds@tsinghua.edu.cn.

codes can significantly reduce storage cost compared with using replication. However, using traditional erasure codes, like Reed-Solomon codes, increases the consumption of network traffic and disk I/O tremendously when systems recover data, and thus will result in high latency of degraded reads. Noticed that data access frequency is Zipf distribution, most data accesses are applied in a small fraction of data, while most data's accessing frequencies are very low. According to this data access skew, this paper presents a mixing storage scheme to solve the high degraded read latency problem of erasure-coded storage systems. In this scheme, two erasure codes are used. A code which has the characteristic of low recovery cost, like Local Reconstruction Code, is used to store frequently accessed data (hot data). Another code which guarantees minimum storage cost, like Hitchhiker code, is used to store infrequently accessed data (cold data). Because hot data occupy most data accesses, most degraded reads should be applied on hot data, then this scheme can have an advantage of low recovery cost from the perspective of the entire system. In the meantime, cold data dominate the whole system in data amount's aspect, then this scheme can have a low storage overhead since cold data are stored by a code that guarantees minimum storage cost. However, there must be some situations in which hot data become cold or cold data become hot. Therefore the mixing storage scheme should be deployed with a code switching procedure. An inappropriate code switching procedure may raise a huge amount of transformation, which is unacceptable. In order to prevent this defect, this paper presents an efficient switching algorithm between Local Reconstruction Code and Hitchhiker code. With this algorithm, the switches between hot data and cold data will not become a bottleneck in the system. While choosing the codes carefully and deploying the efficient code switching algorithm, the mixing storage scheme that this paper presents can be a flawless scheme compared to other existing mixing storage schemes. Through experimental verification, using this storage scheme can reduce 55.8% of degraded read latency compared to traditional Reed-Solomon code. Moreover, when applying this switching algorithm, latency of switching can be reduced to 13.4% and 33.1% respectively. Also, when switching Local Reconstruction Code to Hitchhiker code, which will be the most common situation, the amount of data transferred can be reduced to 10% of the original algorithm by the newly presented code switching algorithm.

Keywords erasure code; degraded read; code switch; fault tolerance; storage

1 引 言

在构建大规模的分布式存储系统时,考虑到成本因素,通常会更倾向于采用低廉但相对容易发生故障的组件.然而,各种故障、错误都可能会导致系统在一段时间内无法正常地访问到需要读取的数据.在Facebook的一个存储容量为数百PB的集群中,每天有大于50次机器不可用时间超过15分钟的情况[1].面对如此频繁不断的机器不可用情况,系统需要保证这些情况不会直接引发数据的不可用,即它需要保证数据的存储是可靠且耐用的.因此,系统需要将存放的数据进行冗余,使得系统能够容忍一定限度的错误.

在分布式存储系统中对数据进行直接复制得到多份副本,是分布式存储系统进行冗余的典型方法[2-3].然而多副本技术的部署会将所有数据都复制多份,这将使存储开销成倍地增加.在数据量迅猛增长的今天,多倍的存储开销将使本已十分昂贵的存储成本变得更高.相较而言,纠删码技术在提供同样的容错能力下能极大地降低存储开销.由此,许多大规模分布式存储系统开始使用纠删码技术,而传统的Reed-Solomon(RS)码[4]又是其中较为普遍的选择.

虽然使用纠删码技术进行容错能够极大地降低存储开销,但使用传统的纠删码技术(例如RS码)也有着一定缺陷.在大规模的分布式存储系统中,后台时常进行着对不可用数据的恢复操作[2].由于

一般的纠删码技术不会产生直接的副本,因此在对某一块进行恢复操作时,必然需要读取其所在分组中的其他数据块或冗余块.但它们往往都在不同的服务器中,甚至会分布在不同机架上,这便会导致磁盘读写以及网络传输数据量明显提高.在一个Facebook的PB级集群中,为了恢复不可用数据,每天需要传输的数据量超过了180 TB[1].

数据的恢复操作一般分为两类——重构(reconstruction)和退化读(degraded read).重构指的是数据已在磁盘等存储设备中丢失,需要在恢复数据后将恢复得到的数据重新写入设备中.而在数据中心中,90%的故障只是暂时的,并不会导致数据的永久丢失[5-6].网络环境的问题、临时的关机或重启操作等,都可能引发这种暂时的故障.应对暂时故障时,恢复操作读取到的数据不必写入设备中,这样的读取过程被称为退化读.退化读需要系统及时地将用户所需数据反馈.数据恢复过程中需要读取及下载的数据量的提升,会导致退化读延迟时间过高.本文将试图从降低数据修复时所需要读取并传输的数据量入手,缓解采用纠删码系统的高退化读延迟问题.

大多数采用纠删码的存储系统对数据只采用一种纠删码进行存储.然而,只采用一种纠删码很难在保持低存储开销的前提下降低退化读延迟.最近,一些系统开始采用多种编码方式存储数据.如文献[7-8]中设计的系统,多副本技术和纠删码技术同时被使用;而在HACFS[9]中,属于同一码种但有着不同参数的纠删码被部署到系统中.这些研究的主要思路是要针对数据的特性选择适合它们的编码.例如,对频繁访问的数据(热数据),应该更关注降低其退化读延迟;而对不频繁访问的数据(冷数据),则应该去保证低存储开销.

本文将提出一种通过两种不同的纠删码进行混合存储的策略以降低系统总体的退化读延迟.对被访问频率较高的热数据,将采用低恢复延迟的编码;而对被访问频率较低的冷数据,将采用能够保证MDS(maximum distance separable)特性[10](同等容错能力下存储开销最小)的编码.这样,根据数据访问的偏斜性,系统总体的退化读延迟将随热数据的退化读延迟降低,而系统总体的存储开销也会足够接近冷数据,达到近似MDS的效果.

在使用混合存储策略时,需要注意到数据的冷热是会随时间变化的.典型的情况是,一份数据在刚被写入时会被标记为热数据,而在一定时间后,这份热数据会变为冷数据.若在系统中分别对冷热数据采用不同的编码进行存储,系统必须有一套相应的编码切换过程.因此,在采用混合存储策略的同时,需要有一种高效的编码切换算法.本文根据局部恢复码(Local Reconstruction Code, LRC)[11]与Hitchhiker(HH)码[12]之间的相关性,提出了这两种编码之间的快速切换算法,保证冷热数据的切换不会成为系统中的瓶颈.

作者利用Ceph[13]实现了上述混合存储策略及相应的编码切换算法,并通过测量退化读延迟、重构时间、编码切换时间及所需传输数据量等指标对其进行评估.

2 动 机

2.1 纠删码之间的权衡

在实际的存储系统中,Google ColossusFS采用了(6,3)RS码①,Facebook HDFS采用了(10,4)RS码②,而Microsoft Azure采用了(12,2,2)LRC[11].之所以它们有着不同的选择,是因为使用不同的码种、参数,对存储系统的可靠性、性能、甚至结构等有着重要的影响.

不同的码种有着各自的优劣.LRC在RS码的基础上通过增加存储开销换取更低的恢复延迟,它们间存在存储开销与恢复延迟的权衡.而对于HH码,它在保持MDS特性的基础上较为有效地降低了恢复延迟,但在编码时需要更长时间.对于阵列码,由于其参数选择、容错能力等限制,现阶段较少会被实现在大规模分布式存储系统中.而对于再生码[14](包括最小存储再生(minimum-storage regenerating, MSR)码和最小带宽再生(minimum-bandwidth regenerating, MBR)码),虽然它们能在存储开销及恢复所需带宽的权衡中达到最优,但将其部署到实际存储系统中的难度较大.表1给出了不同码种在不同方面的对比.

可以看到,不同的码种会对系统各个方面的性能产生重要的影响,它们有着各自的优势及劣势.在系统中,只采用一种编码往往只能使部分指标令人感到满意,而在其它指标存在缺陷.由此,可以通

① Storage Architecture and Challenges, http://web.archive.org/web/20160324185413/http://static.googleusercontent.com/media/research.google.com/en/us/university/relations/facultysummit 2010/storage_architecture_and_challenges.pdf 2010-7-29

② HDFS and Erasure Codes (HDFS-RAID), http://hadoopblog.blogspot.com/2009/08/hdfs-and-erasure-codes-hdfs-raid.html 2009-8-28

表1　不同码种在不同方面的对比

码种	RS码	LRC	HH码	阵列码	MSR码	MBR码
存储开销	最优	较大	最优	一般最优	最优	比较小
参数选择	任意	任意	任意	受限较大	一定受限	任意
恢复延迟	高	低	较低	较高	很低	最低
其它缺点	—	—	编码用时较长	容错能力受限	具体实现复杂	具体实现复杂

过采用两种(或更多种)不同的编码方案,各数据文件可以根据其具体特性选择某一种进行编码,使得系统在整体上,能够综合不同编码的优点,避免各自的缺陷.

本文将在提出的存储策略中选择LRC和HH码进行编码,试图使系统能够综合LRC恢复延迟低及HH码拥有MDS特性的优点.

2.2　数据访问的偏斜性

在大规模分布式存储系统中,数据访问常常出现一种显著的偏斜性[15]. 如图1所示,在大规模分布式存储系统中,数据访问频率呈Zipf分布[15].

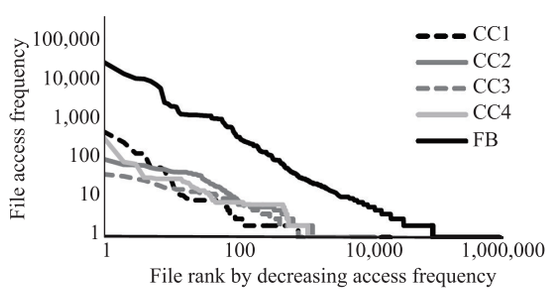

图1　5个HDFS集群的数据访问频率分布图[15],CC{1,2,3,4}表示4个不同的Cloudera上的集群,FB表示Facebook的一个实际集群

在存储系统中,绝大多数的数据访问被应用到很小一部分的数据上,而绝大多数的数据被访问频率极低. 例如当文件总数$N=100000$时,10%的数据将占到约80%的访问量. 一般称被访问频率较高的数据为热数据,而被访问频率较低的数据则被称为冷数据. 这样,热数据占到了系统绝大多数的访问,而冷数据占到了系统绝大多数的数据量.

对于有待解决的退化读高延迟问题,由于热数据占了绝大多数的访问,那么需要进行退化读的绝大多数情况,也将是应用于热数据上. 即,如果能降低热数据的退化读延迟,那么所有数据的退化读延迟在总体上也能降至接近热数据的水平. 同时,由于热数据的数据量在整个系统中处于少数,因此若对热数据提供相对更多的存储空间,对整个系统而言,存储开销也不会有明显的提升. 由此,可以对热数据采用低恢复延迟的编码,而对冷数据采用MDS码取得低存储开销. 这样,整个系统总体的退化读延迟将被热数据所主导,从而取得较低的延迟,提升了性能;而整个系统总体的存储开销将被冷数据所主导,进而降低了成本.

2.3　混合存储策略中的编码切换

在混合存储策略中,数据可能会被存储为两种甚至更多种编码方式. 一份数据当前采用的编码应当根据此刻它的冷热程度决定. 根据时间局部性(temporal locality),系统对其冷或热的判断不会一成不变. 一个典型的例子是,一份数据刚被写入时,系统认为它是热数据;而随着时间推移,越来越多新的数据被写入,系统会认为原来那份数据变为了冷数据. 由热数据变为冷数据应当是系统中常见的情形,但当然,也存在由冷数据变为热数据的可能.

由此,混合存储策略的部署需要数据能在不同编码方式间切换. 最直观的切换方法,是通过读取完整的数据,再进行新编码的编码过程. 然而,读取完整数据的过程会引起大量数据传输. 部署混合存储策略的一大原因是希望通过它降低退化读时的数据传输量,进而降低其延迟. 若是使用上述编码切换算法,系统会不断地在重新编码过程中增加数据传输量,这是难以让人接受的. 因此,混合存储策略需要与一种高效编码切换算法配合使用,避免编码切换时引起的高传输量问题.

判断一个混合存储策略是否存在这一问题,可以通过比较以下两种情形的数据传输量:数据直接被当作冷数据进行编码存储;数据先被当作热数据进行编码存储,之后切换为冷数据的编码形式. 若这两种情形下的数据传输量差距较大,则可以判断此混合存储策略会引起编码切换的高传输量问题.

2.4　当前各混合存储策略的一些缺陷

对于类似文献[7-8]中使用的纠删码和多副本结合的策略,由于存储热数据时使用了多副本技术,因此数据若通过先热后冷的过程,其涉及的数据传输量将至少数倍于直接编码为冷数据的情形,即这些策略会引起编码切换的高传输量问题.

若任意选取两种纠删码进行混合存储,一般而

言将只能通过上一节中提到的直接的重新编码方法进行编码切换，也会引起编码切换的高传输量问题.

在文献[9]提出的HACFS中，作者通过选取同一码族中不同参数的纠删码组成混合存储策略，避免了编码切换的高传输量问题. 然而，由于选取了同一码族的编码，该码族的既定缺陷将难以避免. 如采用product code(PC)时，其编码特性使得这一策略无论如何最多容3错；而不论是采用PC还是LRC，其存储冷数据时所采用编码均不是MDS的，在存储开销方面存在缺陷. 本文工作将尝试选取来自不同码族的两种编码组成混合存储策略.

本文通过选取LRC存储热数据，而使用HH码存储冷数据，并提出它们之间的高效切换算法，组成了一种新的混合存储策略，能够避免上述提及的所有缺陷.

3 混合存储策略

如第2节所述，对热数据，使用恢复延迟更低的编码能降低系统总体的退化读延迟；而对冷数据，采用MDS码能降低系统总体的存储开销. 数据刚进入系统时，将默认被标记为热数据. 系统会对数据进行周期性的检查，当一个热数据一定时间内访问次数没有达到一个设定值，它可能会被标记为冷数据；而当一个冷数据一定时间内的访问次数达到了这个值，它便可能会被标记为热数据. 冷热数据标记的切换也涉及到编码的切换，因此，仅考虑两种编码各自特性是不够的，还需要配置高效的编码切换算法，这一算法将在第4节详细叙述.

注意到恢复延迟低的LRC和拥有MDS特性的HH码的构造均直接地利用了RS码，它们有机会能够进行高效编码切换. 由此，本文提出采用LRC存储热数据，采用HH码存储冷数据的方案. 在下面的叙述中，k表示一个条带中数据块的块数，而m表示一个条带中冗余块的块数.

3.1 局部恢复码(LRC)

LRC在RS码的基础上通过增加局部冗余块以降低恢复延迟. 在LRC中，m个冗余块被细分为全局冗余块及局部冗余块，两者的块数是LRC中的重要参数. 记一个LRC编码中全局冗余块的块数为g，局部冗余块的块数为l，就有$m=g+l$；这样的一个LRC被记为(k,l,g)LRC. 在进行(k,l,g)LRC的编码操作时，首先将k个数据块通过一个(k,g)RS码编码得到g个全局冗余块，之后将k个数据块(相对)平均地分为l个组，每一组将对应一个局部冗余块. 在每一组中，局部冗余块将通过这一组中的数据块直接进行异或运算得到.

局部冗余块的设置减少了恢复数据时需要访问的节点数. 当一个数据块需要恢复时，只需要获取它所在分组对应的局部冗余块及分组中其余的数据块，便可以对它们进行异或运算得出恢复结果. 一个(k,l,g)LRC恢复一个数据块需要另外k/l块，而对应的一个(k,g)RS码则需要k块，是前者的l倍.

图2给出了一个(6,2,2)LRC的例子，其中p_0和p_1是两个全局冗余块，而6个数据块被分为两组，即(x_0, x_1, x_2)及(y_0, y_1, y_2)，它们分别对应的局部冗余块为p_x和p_y. 当x_0需要被恢复时，只需要获取x_1、x_2和p_x三块，即可通过$x_0=x_1 \oplus x_2 \oplus p_x$求得$x_0$.

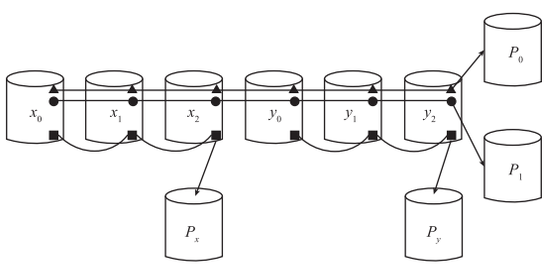

图2 (6,2,2)LRC(例)[11]

3.2 Hitchhiker(HH)码

HH码是由Rashmi等人提出的一种RS码的改进，它的构造是建立在RS码以及同样由Rashmi等人提出的Piggybacking框架[16]的基础上的. 在Piggybacking框架中，传统纠删码中的两个条带会被视作同一条带中的两个子条带. 其核心思想是将其中一个子条带额外的冗余信息与另一子条带的块进行异或运算. 令$a=\{a_i\}$和$b=\{b_i\}$分别表示两个子条带原有的子数据块，$f_i(x)$表示RS码对条带x编码得到冗余块i对应的线性变换函数，$g_i(a)$则表示需要附加到子条带b中的子条带a的冗余信息，在数学上它们也是一种线性变换函数.

HH码将RS码应用在Piggybacking框架中，并将框架中的前$(k+1)$个$g_i(a)$置为0. 而剩下的$(m-1)$个$g_i(a)$，则是将子条带a的k个数据块(相对)平均地分为$(m-1)$组，每组对应一个$g_i(a)$，该$g_i(a)$的计算方法就是将这组内所有数据块进行异或运算.

在丢失一个数据块时，设其在a中的子数据块在上述分组中对应$g_i(a)$. 首先通过子条带b中可用的另外$(k-1)$个数据块以及$f_1(b)$，可以恢复得到丢

失的在b中的子数据块,进而可以求出$f_{(t-k)}(b)$. 而a中子数据块则可以通过上述分组中同一组内其余的a中子数据块,子条带b的第t个子块,以及刚刚求出的$f_{(t-k)}(b)$进行异或运算得到. 这样,要恢复一个数据块,需要读取的子数据块数为$(k+k/(m-1))=km/(m-1)$,折合为$km/(2(m-1))$块,相比于RS码需要k块,在$m\geqslant 3$时均有较为明显的减少.

图3展示了一个(10,4)RS码应用于Piggybacking框架的例子,图4给出了图3对应的HH码. 在图4中,如果要恢复第一个数据块,即a_1和b_1,可以先利用b_2、b_3、…、b_{10}和$f_1(b)$恢复b_1,进而求得$f_2(b)$. 在读取a_2、a_3以及子条带b中的第12个子块$f_2(b)\oplus a_1\oplus a_2\oplus a_3$后,进行异或运算即可恢复得到$a_1$.

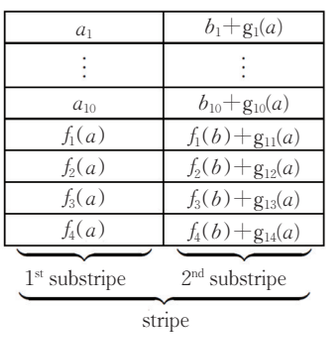

图3　(10,4)RS码应用于Piggybacking框架中[12]

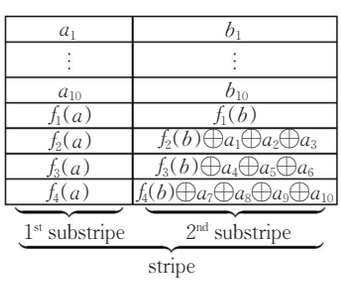

图4　(10,4)Hitchhiker码(例)[12]

在以上恢复过程中,一共需要读取13个子数据块,折合为6.5个数据块,相对于RS码需要10个数据块,需要读取并下载的数据量有了明显的减少.

3.3　参数选择

在确定了储存冷热数据所使用的编码后,需要确定它们的具体参数. 为了满足第4节中编码切换算法的要求,本文提出的混合存储策略将使用(k,t)HH码与$(k,t-1,t)$LRC. 这样的选取貌似有一定限制,但事实上,参数k和t均是可以在合理的范围内任意选择的. 即,在整个存储策略中,占据绝大多数存储量的冷数据所采用的参数是足够灵活的,被限制的仅仅是选取了冷数据编码后的热数据编码,这应当能满足大多数对参数灵活性的需求.

3.4　性能分析

本节将从“退化读开销”和平均失效时间(mean-time-to-failure, MTTF)两个维度对混合存储策略进行分析. 在进行分析时,将使用(k,t)HH码与$(k,t-1,t)$LRC组成混合存储策略,其中参数k和t一般是任意不小于3的正整数.

3.4.1　退化读开销

由于磁盘读写及网络带宽的限制,在存储系统执行数据恢复操作时,其瓶颈往往在于获取恢复过程中所需各块所用的时间. 因此,可以将恢复一块所需要获取的其它块数视作数据恢复的开销. 其中,退化读的开销专注于需要恢复的块是数据块的情况,而重构开销则综合考虑了数据块及冗余块被恢复的情形.

下面作者将对应用本文提出策略的存储系统的退化读开销进行分析. 为简化模型,首先作出一些合理的假设:假设系统中各存储文件的大小固定,且文件被访频率严格地按照Zipf分布. 在叙述中,记N为系统文件总数,p为其中热数据的占比. 为方便叙述,本文将存储开销定义为数据实际存放空间大小与其本身大小之比,将退化读开销定义为恢复一块数据块所需要获取的块数.

可以很容易地计算出LRC和HH码的退化读开销及存储开销. 一个$(k,t-1,t)$LRC的退化读开销为$k/(t-1)$,存储开销为$(1+(2t-1)/k)$;而一个(k,t)HH码的退化读开销为$kt/(2(t-1))$,存储开销为$(1+t/k)$. 这样,当$t\geqslant 3$时(否则HH码并没有对RS码进行优化),$(k,t-1,t)$LRC在退化读开销上仅为(k,t)HH码的$2/t$,而(k,t)HH码在存储开销上较$(k,t-1,t)$LRC少了$(t-1)/k$.

通过计算,可以得到退化读开销—存储开销图. 图5为$N=100000$且$(k,t)=(12,4)$的情况. 需要指出的是,无论k和t如何选取,只要N是确定的,这样一条曲线的形状便不会改变,只会随曲线两个端点的位置(这两个位置被k和t的取值决定)的变化进行伸缩.

图5曲线中加粗部分上的点,它们的存储开销接近于左上方端点的存储开销(即(k,t)HH码的存储开销$(1+t/k)$),而它们的退化读开销则接近于右下方端点的退化读开销(即$(k,t-1,t)$LRC的退化读开销$k/(t-1)$). 即,当p的取值使得系统的退化读开销及存储开销落入这一区域时,系统在整体上获得了HH码MDS特性带来的低存储开销优势,

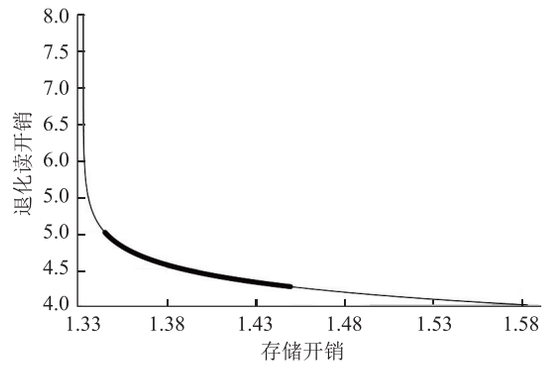

图5 退化读开销—存储开销图

同时获得了LRC局部性带来的低退化读开销优势.

选取参数$(k,t)=(12,4)$,并假设系统中有$N=100000$个文件. 设定系统整体的存储开销在1.4左右(事实上当前情况下实际部署了纠删码的存储系统一般的存储开销为1.33至1.5倍之间[11]). 通过计算,取$pN=26667$能使存储开销最接近1.4. 此时系统总体的退化读开销为4.44.

图6给出了存储开销同为1.4时不同存储策略的退化读开销. 需要指出的是,为了使LRC的容错能力与其它策略接近,作者选取了(12,2,3)LRC,它的存储开销为1.42,略高于另外3种策略.

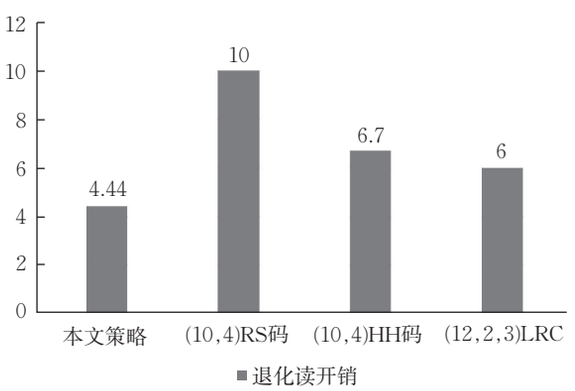

图6 存储开销同为1.4时不同存储策略的退化读开销

综合这4种方案,本文策略显著地降低了退化读开销,进而能够降低退化读延迟.

3.4.2 平均失效时间(MTTF)

同样地,在分析MTTF时,选择参数为$(k,t)=(12,4)$,且系统中有$N=100000$个文件,同时选取$pN=26667$使得存储开销最接近1.4.

作者首先通过马尔可夫模型计算(12,4)HH码和(12,3,4)LRC的MTTF. 按照文献[11]中给出的计算MTTF的经典参数,得到(12,4)HH码的MTTF为3.4×10^{13}年,(12,3,4)LRC的MTTF为3.5×10^{14}年. 由此,根据两种编码所占文件数的比例26667:73333,整个系统的MTTF为2.7×10^{14}年. 表2列出了一些编码(包括多副本)在同样系统参数下计算的MTTF.

表2 不同编码策略的MTTF

编码策略	MTTF(年)
(10,4)RS码	6.7×10^{13}
(10,4)HH码	9.1×10^{13}
(12,2,3)LRC	3.3×10^{13}
3副本	3.5×10^{9}
4副本	7.7×10^{13}
本文策略	**2.7×10^{14}**

由此可见,本文提出策略在实际中可以取得较大的MTTF,即拥有较强的容错能力. 在部署纠删码的实际系统中,一般其容错能力需要超过3副本,而本文提出策略的容错能力更是超过了4副本. 相较于其它单一编码,在参数选择相近时,本文提出策略的容错能力也存在明显的优势.

4 HH码和LRC之间的高效切换算法

4.1 切换算法

注意到HH码和LRC均为RS码的一种改进. 在使用(k,m)HH码进行编码时,首先需要进行(k,m)RS码的编码,再进行一系列Piggybacking框架中关于$g_i(a)$的异或运算;而在使用(k,l,g)LRC进行编码时,g个全局冗余块需要通过执行(k,g)RS码的编码函数得到,而l个局部冗余块则只需要进行异或运算得到. 另外,(k,m)HH码中$g_i(a)$的计算,需要将k个数据块分为$(m-1)$组,而(k,l,g)LRC中局部冗余块的计算,则需要将k个数据块分为l组. 由此可见只要当$k_{HH}=k_{LRC}$且$m_{HH}=g_{LRC}$且$m_{HH}-1=l_{LRC}$时,相应的HH码和LRC能够共用同一个RS码的冗余块,且HH码中作为额外冗余的$g_i(a)$和LRC中该子条带a对应的局部冗余子块应当是等价的. 在这样的设置下,HH码和LRC之间的切换应当是高效的,本文提出的存储策略也将采用这样的参数设置.

下面对实际的切换算法进行介绍. 为避免混淆,在此切换算法中,记$k=k_{HH}=k_{LRC}$及$t=m_{HH}=g_{LRC}$,则$l_{LRC}=m_{HH}-1=t-1$. 即系统中所采用的两种编码分别为(k,t)HH码和$(k,t-1,t)$LRC. 由于HH码会将原RS码中每两个条带视为一个条带中的两个子条带,因此此切换算法也对LRC进行

类似处理,即每一个条带中都包含两个子条带.在LRC中,一个条带中的两个子条带分别进行各自的编码;而在HH码中,其中一个子条带(以下称为子条带a)的额外冗余信息($g_i(a)$)会被异或到另一子条带(以下称为子条带b)的冗余子块中.

从$(k,t-1,t)$LRC转换为(k,t)HH码时,首先将子条带a中的各局部冗余子块读取并传输到对应的全局冗余块所在节点,并与该节点中子条带b的全局冗余子块进行异或运算,之后再将各局部冗余块删除即可.

从(k,t)HH码转换为$(k,t-1,t)$LRC时,首先计算出所有局部冗余块,之后再将其中子条带a中的局部冗余子块传到对应的全局冗余块所在节点,并与该节点中子条带b的全局冗余子块进行异或运算,使得这些子条带b的全局冗余子块不再包含子条带a中的额外冗余信息$g_i(a)$.

图7和图8给出了当$(k,t)=(6,3)$时,(6,2,3)LRC与(6,3)HH码间切换的流程示意图.

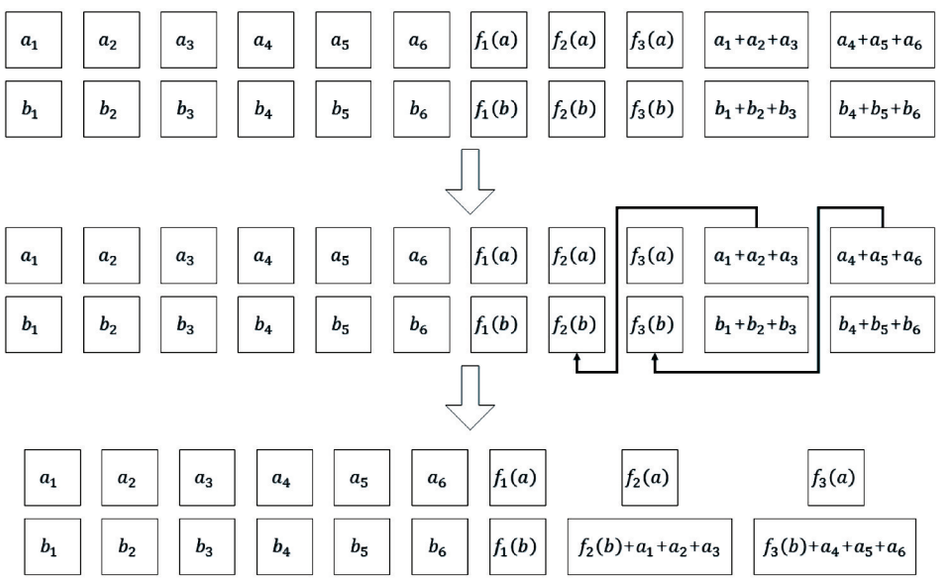

图7　将(6,2,3)LRC切换为(6,3)HH码

4.2　算法效果分析

在系统需要将热数据转换为冷数据时,会将$(k,t-1,t)$LRC切换为(k,t)HH码.若采用重新编码算法,在重新编码前需要获取全部数据块,并在编码后再将新的冗余块发送到各个节点,因此至少需要传输$(k+t-1)$块.若采用本文提出的算法,只需要传输$(t-1)$个子块,即$(t-1)/2$块.当$k=12$且$t=4$时,本文提出的算法在数据传输量方面只需要重新编码算法的1/10.

同时,本文提出的算法是可以很自然地分布式地进行的,这将对编码切换延迟的降低很有帮助.

在系统需要将冷数据转换为热数据时,会将(k,t)HH码切换为$(k,t-1,t)$LRC.若采用重新编码算法,至少需要传输$(k+2t-2)$块.若采用本文提出的算法,需要传输$(2k+t-1)$个子块,即$(2k+t-1)/2$块.当$k=12$且$t=4$时,本文提出的算法降低了重新编码算法数据传输量的1/4.

同样地,本文提出的算法此时也可以很自然地分布式地进行,由此若只考虑编码切换延迟,应当可以降为重新编码算法的$1/(t-1)$,当$k=12$且$t=4$时,此延迟能降为重新编码算法的1/3.

综合上述两种切换过程,可以看出本文提出的高效切换算法相较于重新编码算法有着明显的优化.值得注意的是,一般而言,由于系统中数据一般会不断增加,且数据访问有着时间局部性,因此热数据变为冷数据的情况应当会比冷数据变为热数据的情况多得多.又由于本文提出的切换算法在热数据变冷时能够取得更明显的优化效果,因此总体上的性能优化是相当显著的.

而在计算效率方面,本文提出算法只需要进行XOR运算,且涉及范围相对整一组数据较小,这对比于重新编码算法需要整份数据进行的伽罗瓦域运算也有了明显优化.虽然计算时间并不是编码切换的瓶颈所在,但计算效率的大幅提升,能有效地帮助系统节省计算资源,从而能为上层实际应用分配更多计算资源,进一步提升系统总体性能.

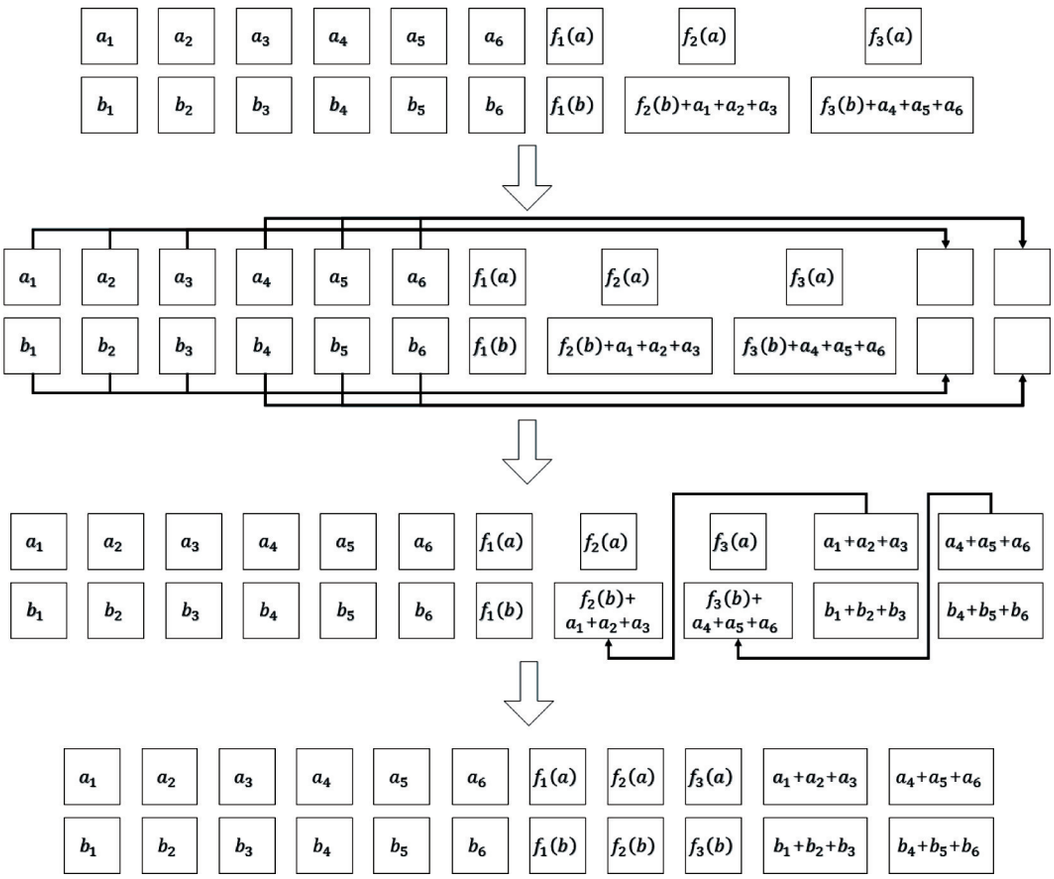

图8 将(6,3)HH码切换为(6,2,3)LRC

5 实 验

在本节中,作者通过在Ceph集群中进行实验,对本文提出的存储策略在数据恢复延迟及编码切换时间上进行实际评估.

实验选取参数$(k,t)=(12,4)$,即在系统中部署(12,3,4)LRC和(12,4)HH码两种编码.根据这一参数选择,在Ceph系统中需要有至少19个同级存储单位.实验中采用4台服务器,每台服务器中均能提供7块2 TB硬盘作为7个OSD,这样整个系统共有28个OSD且存储总量可达50 TB.各服务器间通过一个局域网进行互连.Ceph系统版本为Ceph v10.2.7 Jewel LTS.本文存储策略的实现利用了C语言进行程序编写,在编码及解码时使用Jerasure 2.0[17]进行RS码的编解码,并在其基础上实现了HH码及LRC的编解码以及编码切换.数据恢复延迟是通过将一部分OSD标记为不可用进行测量的.

实验中将存储$N=100000$个大小为256 MB的文件,系统总的存储开销设定为1.4,总的存储量应约为35 TB.由于选取参数k为12,因此每一数据块大小约为21.3 MB.实验中数据访问是根据Zipf分布模拟的.

5.1 数据恢复延迟

通过实验,测得利用(12,4)HH码恢复数据块用时1198 ms,恢复冗余块用时1755.2 ms;而利用(12,3,4)LRC恢复数据块或局部冗余块需要用时582.2 ms,恢复全局冗余块用时1763.4 ms.这样,系统总体的退化读平均延迟为649.5 ms,平均重构时间为1184.6 ms.

作为对比,实验还测量了3.4节中提及到的三种存储开销均为1.4的编码的退化读延迟和重构时间,如图9所示,其中数据均折合为每需要恢复1 MB所需要时间.

由此可见,在本文最为关注的退化读延迟一项中,本文提出的策略相较RS码降低了55.8%,相较HH码降低了38.0%,相较于存储开销稍高的LRC也降低了25.7%.而在重构时间方面,本文提出的策略相较RS码降低了19.3%,与同样存储开销而MTTF略小的HH码基本一致.这说明了,本文提出的策略能够在保持高容错能力、参数选择灵活性、合理的存储开销等前提下,大大降低退化读延迟,且

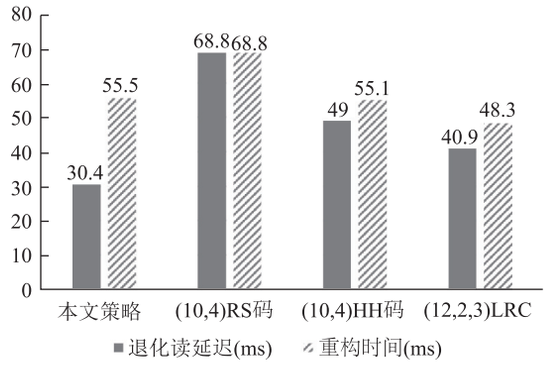

图9　不同策略恢复1 MB数据的用时

对重构时间进行了一定的优化.

5.2　编码切换延迟

在实验中可以实现本文提出的高效编码切换算法以及重新编码算法. 图10给出了对一个文件(256 MB)在两种切换情形下应用不同算法分别需要的时间.

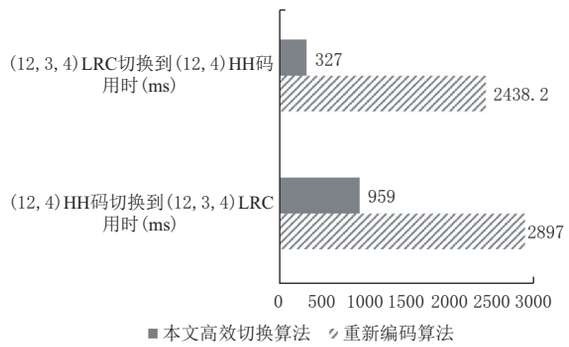

图10　两种切换情形下应用不同切换算法的用时

从实验结果可以得出,当从(12,3,4)LRC切换至(12,4)HH码时,利用本文提出的算法用时约为重新进行HH码编码用时的13.4%. 与数据恢复类似,编码切换的耗时应取决于数据的网络传输延迟. 根据在4.2节中的分析,利用本文提出的算法所需数据传输量应为重新编码算法的10%,这一数值与用时的比值13.4%确实颇为接近.

而当从(12,4)HH码切换至(12,3,4)LRC时,利用本文提出的算法相较重新进行LRC编码用时能够减少为33.1%,与4.2节中分析得到的33.3%基本相符.

通过实验,可以证明本文提出的编码切换算法能够十分有效地降低切换用时,使得本文提出的存储策略在需要进行冷热数据切换时不会产生较高延迟进而形成瓶颈.

5.3　编码切换数据传输量

本节将通过实验,测出数据直接被当作冷数据进行编码存储所需要的数据传输量,及数据先被当作热数据进行编码存储,之后再切换为冷数据的编码形式所需要的数据传输量,对它们进行比较,以验证本文提出的编码切换算法能够使得本文的混合存储策略在编码切换时避免引起高数据传输量. 图11首先给出了一个文件(256 MB)直接编码为(12,4)HH码在系统中的数据传输量,同时给出了使用本文策略,先编码为(12,3,4)LRC的数据传输量及之后切换为(12,4)HH码的切换传输量. 另外,为了进行不同混合存储策略间的比较,图11中还给出了不使用本文提出的编码切换算法的(12,3,4)LRC和(12,4)HH码混合时的情形,以及三副本和(12,4)HH码混合时的情形.

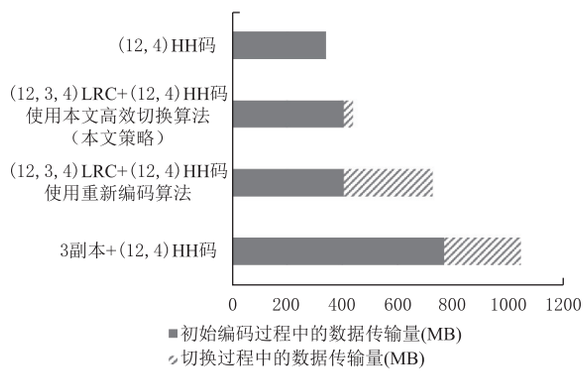

图11　不同混合存储策略下切换传输量的比较

可以通过计算以下两种比值进行评估. 比值I是切换过程的数据传输量比初始编码过程的数据传输量,它可以表示切换算法的效率,此比值越接近0则效率越高. 比值II是数据经过初始编码再切换这两个过程总的数据传输量比数据被直接编码成为热数据形式的数据传输量,它可以表示混合存储策略是否引起了在切换编码时的高数据传输量问题,若此比值接近1则说明不会引起这一问题. 表3给出了图11对应的比值I和比值II,并加入了与HACFS(包括使用两个不同PC的HACFS-PC及使用两个不同LRC的HACFS-LRC)的比较.

从表3中可以看出,本文策略的比值I和比值II

表3　不同策略的比值I和比值II

编码策略	比值I	比值II
(12,3,4)LRC + (12,4)HH码	0.790	2.125
3副本 + (12,4)HH码	0.361	3.063
HACFS-PC	0.333	1.714
HACFS-LRC	0.300	1.625
本文策略	**0.079**	**1.281**

都足够小,这说明了本文提出的编码切换算法足够高效,且不会在切换编码时引起高数据传输量.值得一提的是,本文策略的比值I和比值II均明显较HACFS要小.虽然本文策略选用了两种属于不同码族的编码,但由于高效编码切换算法的提出,本文策略在编码切换方面取得了更好的效果.

6 相关工作

Xia等人在文献[9]中首次提出在同一系统中部署不同纠删码的想法,实现了在系统中部署两种不同参数的PC或两种不同参数的LRC.然而,文献[9]只考虑在同一码种中选择不同参数的编码,未能充分利用不同编码各自的优点.本文工作首次实现了在系统中部署两种属于不同码种的编码.

另外,本文工作属于提高纠删编码系统恢复性能的范畴.近年来,关于这一课题的最主流研究主要立足于两种不同编码:再生码及LRC.

Dimakis等人在文献[14]中首次提出了再生码的模型,再生码能够在进行数据恢复时使得需要传输的数据量最小化.文献[18-24]等研究都对再生码进行了构造.目前,能将再生码部署到实际系统中的研究较为少见.在文献[25]中,一种较为特殊的MSR码被首次实现在HDFS和Ceph中.最近,文献[26]的作者将他们提出的目前最为高效的clay code实现到Ceph中.由于有理论支撑,再生码的恢复性能是单一编码中最优的,甚至优于很多情况的混合存储策略.但在采用混合存储策略时,也可以通过采用退化读开销及存储开销差距较大的两种编码进行混合,通过数据访问的偏斜性,在退化读性能方面超越再生码.比如,一个(10,4)MSR码的退化读开销最小为3.25;而采用(12,4)HH码及(12,6,4)LRC的混合存储策略,在$N=100000$且存储开销设定为同样的1.4时,其退化读开销仅约为3.值得一提的是,根据本文提出切换算法的思想,也能找到一种(12,4)HH码和(12,6,4)LRC之间的高效切换算法.鉴于再生码的恢复性能是单一编码中最优的,如何将再生码实用地应用到类似的混合存储策略中也是作者下一步的工作.

在恢复数据时减少需要连接的节点数并减少相应的数据传输量,可以降低数据恢复延迟.据此,有研究[27-30]提出了一类满足上述特性但不再满足MDS的编码,即LRC.它们对LRC进行了理论方面的研究,给出了许多重要的结论及参数选择条件.而在文献[11,30]中,不同的LRC成功地被部署在实际系统中.本文所使用的LRC便是文献[11]中部署的编码.

7 结束语

本文根据分布式存储系统中文件访问的偏斜性,提出对数据冷热进行划分的混合存储策略.如此,整个系统的退化读延迟接近于热数据,可通过低恢复开销的编码LRC得到性能优化;而整个系统的存储开销更接近于冷数据,可通过低存储开销的编码HH码降低存储开销.本文提出的这一策略也是第一个成功地采用两个不同码族编码的混合存储策略.

通过分析,本文提出的这一存储策略在其它重要指标基本不变的情况下,能使系统的退化读延迟得到大幅降低.作者还通过搭建具体的系统对此策略进行实验,部署了(12,3,4)LRC和(12,4)HH码两种编码.实验结果证明,此策略十分有效地降低了退化读延迟.

为了解决上述存储策略中不同编码间切换的高延迟、高数据传输量问题,本文提出了一种LRC和HH码之间的高效切换算法,并将其部署到实验系统中.实验结果证明,本文提出的这一高效切换算法,能够有效降低编码切换延迟及数据传输量,保证了本文提出的存储策略在实际系统中部署的可用性.

参 考 文 献

[1] Rashmi K V, Shah N B, Gu D, et al. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: a study on the Facebook warehouse cluster//Proceedings of the 5th USENIX Workshop on Hot Topics in Storage and File Systems. San Jose, USA, 2013: 8

[2] Ghemawat S, Gobioff H, Leung S. The Google file system. ACM SIGOPS Operating Systems Review, 2003, 37(5): 29-43

[3] Shvachko K, Kuang H, Radia S, et al. The Hadoop distributed file system//Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). Lake Tahoe, USA, 2010: 1-10

[4] Reed I S, Solomon G. Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics, 1960, 8(2): 300-304

[5] Ford D, Labelle F, Popovici F I, et al. Availability in globally distributed storage systems//Proceedings of the 9th USENIX Symposium on Operating Systems Design and

Implementation. Vancouver, Canada, 2010: 61-74

[6] Khan O, Burns R, Plank J, et al. Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads//Proceedings of the FAST' 12: 10th USENIX Conference on File and Storage Technologies. San Jose, USA, 2012: 251-264

[7] Ma Y, Nandagopal T, Puttaswamy K P N, et al. An ensemble of replication and erasure codes for cloud file systems//Proceedings of the 32nd IEEE International Conference on Computer Communications. Turin, Italy, 2013: 1276-1284

[8] Friedman R, Kantor Y, Kantor A. Replicated erasure codes for storage and repair-traffic efficiency//Proceedings of the 14-th IEEE International Conference on Peer-to-Peer Computing. London, UK, 2014: 1-10

[9] Xia M, Saxena M, Blaum M, et al. A tale of two erasure codes in HDFS//Proceedings of the 13th USENIX Conference on File and Storage Technologies. Santa Clara, USA, 2015: 213-226

[10] Roth R M, Lempel A. On MDS codes via Cauchy matrices. IEEE Transactions on Information Theory, 1989, 35(6): 1314-1319

[11] Huang C, Simitci H, Xu Y, et al. Erasure coding in Windows Azure storage//Proceedings of the 2012 USENIX Annual Technical Conference. Boston, USA, 2012: 15-26

[12] Rashmi K V, Shah N B, Gu D, et al. A "Hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers//Proceedings of the 2014 ACM Conference on Special Interest Group on Data Communication. Chicago, USA, 2014: 331-342

[13] Weil S A, Brandt S A, Miller E L, et al. Ceph: a scalable, high-performance distributed file system//Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation. Seattle, USA, 2006: 307-320

[14] Dimakis A G, Godfrey P B, Wu Y, et al. Network coding for distributed storage systems. IEEE Transactions on Information Theory, 2010, 56(9): 4539-4551

[15] Chen Y, Alspaugh S, Katz R. Interactive analytical processing in big data systems: a cross-industry study of MapReduce workloads. Proceedings of the VLDB Endowment, 2012, 5(12): 1802-1813

[16] Rashmi K V, Shah N B, Ramchandran K. A piggybacking design framework for read-and download-efficient distributed storage codes//Proceedings of the 2013 IEEE International Symposium on Information Theory. Istanbul, Turkey, 2013: 331-335

[17] Plank J S, Greenan K M. Jerasure: a library in C facilitating erasure coding for storage applications version 2.0. Knoxville, USA: University of Tennessee, Technical Report: UT-EECS-14-721, 2014

[18] Shah N B, Rashmi K V, Kumar P V, et al. Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff. IEEE Transactions on Information Theory, 2012, 58(3): 1837-1852

[19] Shah N B, Rashmi K V, Kumar P V, et al. Interference alignment in regenerating codes for distributed storage: necessity and code constructions. IEEE Transactions on Information Theory, 2012, 58(4): 2134-2158

[20] Rashmi K V, Shah N B, Kumar P V. Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction. IEEE Transactions on Information Theory, 2011, 57(8): 5227-5239

[21] Tamo I, Wang Z, Bruck J. Zigzag codes: MDS array codes with optimal rebuilding. IEEE Transactions on Information Theory, 2013, 59(3): 1597-1616

[22] Cadambe V R, Huang C, Li J. Permutation code: optimal exact-repair of a single failed node in MDS code based distributed storage systems//Proceedings of the 2011 IEEE International Symposium on Information Theory. Saint Petersburg, Russia, 2011: 1225-1229

[23] Ye M, Barg A. Explicit constructions of optimal-access MDS codes with nearly optimal sub-packetization. IEEE Transactions on Information Theory, 2017, 63(10): 6307-6317

[24] Sasidhara B, Vajha M, Kumar P V. An explicit, coupled-layer construction of a high-rate MSR code with low sub-packetization level, small field size and $d < (n-1)$// Proceedings of the 2017 IEEE International Symposium on Information Theory. Aachen, Germany, 2017: 2043-2047

[25] Pamies-Juarez L, Blagojevic F, Mateescu R, et al. Opening the chrysalis: on the real repair performance of MSR codes// Proceedings of the 14th USENIX Conference on File and Storage Technologies. Santa Clara, USA, 2016: 81-94

[26] Vajha M, Ramkumar V, Puranik B, et al. Clay codes: moulding MDS codes to yield an MSR code//Proceedings of the 16th USENIX Conference on File and Storage Technologies. Oakland, USA, 2018: 139-154

[27] Gopalan P, Huang C, Simitci H, et al. On the locality of codeword symbols. IEEE Transactions on Information Theory, 2012, 58(11): 6925-6934

[28] Tamo I, Papailiopoulos D S, Dimakis A G. Optimal locally repairable codes and connections to matroid theory. IEEE Transactions on Information Theory, 2016, 62(12): 6661-6671

[29] Tamo I, Barg A. A family of optimal locally recoverable codes. IEEE Transactions on Information Theory, 2014, 60(8): 4661-4676

[30] Sathiamoorthy M, Asteris M, Papailiopoulos D, et al. XORing elephants: novel erasure codes for big data. Proceedings of the VLDB Endowment, 2013, 6(5): 325-336

WANG Zi-Zhong, Master's degree candidate. His research interests include fault tolerance in storage systems.

WANG Hai-Xia, Ph. D. associate professor. Her research interests include computer architecture.

SHAO Ai-Ran, Ph. D. candidate. His research interests include computer architecture.

WANG Dong-Sheng, Ph. D. professor. His research interests include computer architecture.

Background

This paper presents a mixing erasure-coded storage scheme and discusses the high degraded read latency problem in erasure-coded systems, which is a part of the area of fault tolerance in storage systems.

Replication and erasure codes are two typical ways to achieve fault tolerance in distributed storage systems. Compared to replication, using erasure codes can significantly reduce storage cost under the same level of fault tolerance. However, using traditional erasure codes may increase the consumption of network traffic and disk I/O tremendously when systems recover data, and thus will result in higher latency of degraded reads and longer reconstruction time.

To solve this problem, two major kinds of codes, Local Reconstruction Codes (LRCs) and Minimum Storage Reconstruction (MSR) Codes, are widely studied in recent years.

At the same time, some mixing storage schemes are presented. These schemes use at least two codes to store data. According to the data access skew, choosing the codes carefully can be helpful to reduce degraded read latency. The most famous one should be HACFS, which combines two codes in one code family. However, these schemes have obvious defects. Most of them may raise a large amount of transferring data while code switching, and HACFS can be restricted in the only one code family.

The new mixing storage scheme that this paper presents can avoid these defects mentioned above. The mainly breakthrough is that this paper presents an efficient code switching algorithm between two codes in different code families. By this code switching algorithm, there will be just a little fraction of data needed to be transferred when switching codes, while the system uses two different codes, LRC and Hitchhiker code. Storage systems can have advantage in performance by deploying this new scheme rather than the previous ones.

This research is a part of a project which studies the reliability of systems. A mixing storage scheme can be a good solution to achieve fault tolerance and also keep a good performance at the same time.

This work is supported by the National Key Research and Development Program of China (Grant No. 2016YFB1000303) and the Guangdong Province Key Research and Development Program of China (Grant No. 2018B010115002).