第 42 卷 第 6A 期
2015 年 6 月

计 算 机 科 学
Computer Science

Vol. 42 No. 6A
June 2015

一种面向云存储数据容错的 ARC 缓存淘汰机制

伍秋平[1] 刘 波[1] 林伟伟[2]

(华南师范大学计算机学院 广州 510631)[1] (华南理工大学计算机科学与工程学院 广州 510006)[2]

摘 要 Hadoop 默认采用副本冗余方式实现数据容错,但这种容错方式存在着空间占用过大、存储效率低等问题。为此,在分析了 ARC 缓存淘汰算法的基础上,提出了一种面向云存储数据容错的 ARC 缓存淘汰机制 ARCMFF。在文件的访问过程中,ARCMFF 通过维护一个 LRU 队列和一个 LFU 队列统计出访问频率高的文件并将其加入缓存系统中,以提高访问性能;在 ARCMFF 中,大部分文件采用的是纠删码方式容错存储,只有缓存中的文件才用副本冗余方式存储。纠删码的编码效率很高,因此系统能够节省大量的存储空间。实验结果表明,在分布式文件系统中,ARCMFF 能够节省文件存储空间,大大地提高 Hadoop 的存储效率,且能够在一定程度上提高文件的写入性能。

关键词 云存储,数据容错,Hadoop,副本冗余,纠删码,ARC

中图法分类号 TP393 文献标识码 A

Adaptive Replacement Cache Mechanism for Fault Tolerance in Cloud Storage

WU Qiu-ping[1] LIU Bo[1] LIN Wei-wei[2]

(College of Computer, South China Normal University, Guangzhou 510631, China)[1]

(School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China)[2]

Abstract Hadoop adopts replicas as the default data fault tolerance, while this fault tolerance mechanism occupies much storage space and has a low storage efficiency. To solve this problem, this paper proposed an adaptive replacement cache mechanism for fault tolerance in cloud storage, based on analyzing the cache replacement algorithm ARC. When users access the file system, ARCMFF statistics files' access frequency by maintaining a LRU queue and a LFU queue, and then adds those files highly accessed to the cache for better system performance. In ARCMFF, the majority of files are stored by erasure code, while only few files in the cache are stored by replica. The erasure code is highly encoded and has a higher storage efficiency, resulting that the distributed file system can save amount of storage space. According to series of experiments, we approved that the distributed file systems with ARCMFF can save file storage space greatly, improve storage efficiency and achieve higher reading and writing performance.

Keywords Cloud storage, Data fault tolerance, Hadoop, Replica, Erasure code, ARC

1 引言

云计算、物联网和社交网络等新兴服务促使人类社会的数据种类和规模正以爆炸式的速度指数级增长。国外媒体把 2013 年称为“大数据元年”[1]。在大数据时代下,人们对数据的存储要求也越来越高。特别是随着存储系统中存储介质数目的增加以及存储介质的多样化和复杂化,存储系统的存储介质错误[2,3]或者存储介质中的潜在扇区错误 latent sector errors[4]出现的概率也越来越高,海量文件的存储需求和容错需求越来越难以满足。分布式文件系统[5]将大量文件分散存储在各台子节点,并具有容错和故障恢复机制,满足了大数据时代的文件存储的空间需求和容错性需求。

目前在国内外已经有许多种类的分布式文件系统,如 Google 的 GFS[6]、Apache 的 HDFS[7]、Amazon 的 S3[8]及阿里的 TFS[9]等。它们的容错机制都是通过数据冗余来实现的,而数据冗余一般可以通过副本冗余或纠删码[10]方式来实现。文件系统单一地采用副本冗余或纠删码的方式存储文件各有其优缺点[11]。根据两者特点,文献[12]提出了一种动态结合副本冗余与纠删码的容错机制,有助于提高分布式文件系统的存储空间利用率及文件的写入性能。此外,我们之前在云存储的数据副本放置策略[13]、数据副本数的动态变化[14]、数据的负载均衡[15]及数据存储的能耗管理[16]等方面也进行了不少研究。

然而,Hadoop 默认采用副本冗余方式实现数据容错[7,14],但这种容错方式存在着空间占用过大、存储效率低等问题。为此,本文根据缓存算法的思想,提出了一种基于 ARC 缓存算法并能对副本冗余和纠删码进行动态调整的容错机制。该机制将存储系统中的所有文件都采用纠删码存

本文受国家自然基金项目(61402183),广东省科技计划项目(2013B010401005,2013B010401024),广州市科技计划项目(2013Y200077),中央高校基本科研业务费重点项目(2013ZZ0044),华南师范大学研究生科研创新基金资助项目(C1074522)资助。

伍秋平(1991—),男,硕士生,主要研究方向为大数据、云计算,E-mail:cs.qiupingwu@gmail.com;刘 波(1968—),男,博士,教授,主要研究方向为大数据、云计算;林伟伟(1980—),男,博士,副教授,主要研究方向为大数据、云计算(通信作者)。

储,以节省存储空间。在此之外,系统又设计了一个很小的“缓存区域”,缓存区域中的文件采用副本冗余方式存储,以提高文件访问性能。通过 ARC 算法进行缓存替换调度,可极大地提高缓存的命中率,从而使系统既能有较好的文件访问速度,又只占用较小的文件存储空间。这种机制能够很好地结合副本冗余和纠删码的优点,使分布式文件系统能够拥有很好的访问和存储性能。

本文第 2 节介绍了一种命中率很高的 ARC[17] 缓存淘汰算法,并对其命中率进行了分析;第 3 节介绍了当前云存储系统数据的容错机制,提到了两种常见的数据冗余方式:副本冗余和纠删码,并分析了它们各自的特点。基于它们各自的优缺点,引出了将文件用纠删码和副本冗余结合起来存储的思想;第 4 节提出了一种面向云存储数据容错的 ARC 缓存淘汰机制 ARCMFF,并解释了它的基本思想、工作机制及讲解了该机制的详细设计。最后通过真实的实验环境,对采用了该机制的分布式文件系统(以 HDFS 为例)进行性能测试,并对实验结果进行了分析。实验结果表明,采用该机制后,分布式文件系统能在很大程度上节省文件的存储空间,并能在一定程度上提高文件的写入性能。

2 Cache 文件系统中的 ARC 算法

2.1 Adaptive Replacement Cache[17]

Adaptive Replacement Cache(ARC)算法是由 Nimrod Megiddo 和 DharmendraS. Modha 提出的一种缓存替换算法。ARC 算法是一种基于 DBL 构建的算法,DBL 算法共维护 $2c$ 个页面,它会从中挑选出来 c 个页面加入缓存中。ARC 算法的基本思想为:假设有两个队列 L_1 和 L_2,分别表示最近访问(LRU)队列和最经常访问(LFU)队列,它们都是根据最近访问时间的先后来排列的,L_1 和 L_2 的容量都为 c。现把 L_1 分为 T_1 和 B_1 两部分,T_1 中的页面比 B_1 中的页面距离上次访问时间更近;把 L_2 分为 T_2 和 B_2 两部分,T_2 中的页面比 B_2 中的页面距离上次访问时间更近。$T_1 \cup T_2$ 中的页面即为 Cache 中的 c 个页面,故 $|T_1|+|T_2|$ 正好为 c。算法设定 T_1 的目标容量为 p,那么 B_1 的目标容量为 $c-p$,T_2 的目标容量为 $c-p$,则 B_2 的目标容量为 p。

在文件的访问过程中,所请求页面可能已经存在于缓存中(也就是 $T_1 \cup T_2$ 中),这时称为该请求缓存命中。如果请求缓存命中,那么将该页面移到 T_2 队首。如果请求未命中,那么就从 T_1 或者 T_2 中挑选出一个页面换出缓存。算法的替换策略如下:

如果 $|T_1|>p$,那么替换 T_1 中的 LRU 页面;

如果 $|T_1|<p$,那么替换 T_2 中的 LRU 页面;

如果 $|T_1|=p$,且所缺页面能在 $B_1(B_2)$ 中找到,那么替换 $T_2(T_1)$ 中的 LRU 页面。

ARC 算法对工作负载的历史具有学习能力。学习的方式如下:如果有一个页面在 B_1 中命中,那么我们应该增大 T_1 的容量;如果有一个页面在 B_2 中命中,说明我们应该增加 T_2 的容量。也就是说,当页面在 B_1 中命中,我们应该增加 p 的值(设增加的值为 δ_1),如果页面在 B_2 中命中,我们应该减少 p 的值(设减少的值为 δ_2)。

其中,

$$\delta_1=\begin{cases}1, & \text{if } |B_1|\geqslant|B_2|\\ \dfrac{|B_2|}{|B_1|}, & \text{otherwise}\end{cases}$$

$$\delta_2=\begin{cases}1, & \text{if } |B_2|\geqslant|B_1|\\ \dfrac{|B_1|}{|B_2|}, & \text{otherwise}\end{cases}$$

ARC 算法的特点:

从上面的算法描述中可以看出,T_1(容量为 p)代表的是最近访问(LRU)页面,而 T_2(容量是 $c-p$)代表的是最经常访问(LFU)页面,由于 p 能够在实际的工作负载中动态地改变,ARC 算法能够很好地结合 LRU 和 LFU 算法。即使是在访问模型经常发生变化的文件系统中,ARC 也能够同时捕获访问序列的 Recency 和 Frequency,故 ARC 算法能够拥有较高的缓存命中率。

3 云存储数据容错技术

存储性能的好坏通常有两个指标:数据的可用性和容错性。可用性意味着系统能够在可接受的时间范围成功地从系统中读取数据。而容错性是指系统不会因为某一节点的突然故障而导致数据丢失,系统拥有一定的数据容错能力。云存储的数据容错机制是通过数据冗余方式来实现的,通常有副本冗余和纠删码两种常用的数据冗余方式。

3.1 副本冗余

副本冗余是指在存储系统中为同一文件创建多个拷贝,以提高数据资源的可用性。副本冗余简单、直观,能有效提高系统的可用性、降低访问延迟、避免热点的产生、实现负载平衡,但其存储空间消耗比较大。通常副本数越多,数据的可靠性越高。但副本数量也不是越多越好,不合理的副本数量和分布方法可能会带来不必要的开销,影响系统的总体性能。副本冗余不涉及专门的编码,数据出错时恢复快、性能较好,但存储利用率极低,当存放 N 个副本时磁盘利用率仅有 $1/N$。尤其是当系统规模很大时副本冗余技术带来的额外存储空间的开销很大,导致存储成本非常高。

3.2 纠删码

纠删码起源于通信传输领域,起初主要是用于解决数据传输中的检错和纠错问题。后来纠删码逐渐应用到存储系统中的数据检错和纠错问题中,以提高存储系统的可靠性,并根据存储系统应用的特点逐步改进和推广[18]。在存储系统中纠删码技术主要是通过纠删码算法将原始的数据块进行编码得到冗余块,并将数据块和冗余块一并存储起来以达到容错的目的。其基本思想是:将 k 块原始的数据元素通过一定的编码计算得到 m 块冗余元素,对于这 $k+m$ 块的元素当其中任意的 k 块元素出错包括数据和冗余出错时,均可以通过对应的重构算法恢复出原来的 k 块数据。基于纠删码的方法与传统的副本冗余技术相比具有冗余度低、磁盘利用率高和容错性较副本冗余强等优点。

3.3 副本冗余与纠删码的结合

副本冗余和纠删码都有其各自的优缺点,那么能不能将副本冗余和纠删码结合起来呢?关于副本冗余和纠删码及其动态结合,已经有不少前人在这方面做了研究,如《Erasure Code Replication Revisited》[19]。

在文件的不断访问过程中,会产生两类数据。

1)热数据:访问频率高,用户经常查看和下载的数据或数据块;

2)冷数据:在用户上传以后,很少访问的数据或数据块。

在文件系统中,可将热数据用副本冗余的方式存储,这样一方面可以保证数据的可用性更高,另一方面也可在文件出错或丢失的情况下,快速恢复文件;将冷数据用纠删码的方式存储起来,可节省大量的存储空间。在文件的访问过程中,热数据和冷数据并不是一成不变的,而是动态变化的。系统可动态调整副本冗余和纠删码方式来存储冷热数据文件。

4 ARCMFF 设计

4.1 基本思想

本文在 ARC 算法的基础上,通过将其应用到分布式文件系统之上,结合副本冗余和纠删码两种数据容错方式,设计出一种面向云存储数据容错 ARC 缓存淘汰机制 ARCMFF(adaptive replacement cache mechanism for fault tolerance)。

在内存与 Cache 页面之间的缓存调度模型中,内存中页面的可用存储空间大,但访问速度慢;而 Cache 中的页面访问速度快,但可用存储空间小。在一个副本冗余和纠删码共存的文件系统中,采用纠删码存储的文件存储利用率高,可用存储空间大,但访问速度慢;采用副本冗余存储的文件访问速度快,但可用存储空间小。因此,可将纠删码存储的文件等效为内存中的页面,副本冗余存储的文件等效为 Cache 中的页面。借鉴缓存调度算法,来动态决定文件的存储容错方式。ARCMFF 就是借鉴这一等效思想,来实现副本冗余和纠删码动态结合存储。

ARCMFF 将云存储文件系统中的区域分为存储区 SA 和缓存区 CA。

存储区(Storage Area,SA),是云存储系统中用于实现文件的基本存储的区域,系统中的所有文件将存储于 SA 中,为了节省存储空间,SA 中的所有文件将采用纠删码方式存储。

缓存区(Cache Area,CA),是云存储系统中用于提高文件访问速度及文件出错时恢复速度而存在的区域,CA 中的所有文件将采用副本冗余方式存储。CA 中文件的存储空间有限,空间大小可由系统管理员设定。

SA 中的文件采用的是纠删码方式存储,故文件的访问速度将略慢于正常文件系统,但较副本冗余方式容错的文件系统来说,节省了大量的存储空间(理论上,当副本数为 3 时,1 单位大小的文件,使用纠删码容错存储时需要占用 1.4 单位的存储空间,而使用副本冗余方式容错存储时则需要占用 4 单位的存储空间),且容错性强于 CA 中的文件。CA 中的文件,由于采用了副本冗余方式存储,访问速度很快,系统出错时恢复快,但占用空间较大。系统中的所有文件都有一份纠删码的拷贝存在于 SA 中,但只有访问频率高的那部分文件存在于 CA 中。

提出的 ARC 缓存淘汰机制的工作流程如下:

(1)用户上传文件,此时文件是以纠删码形式存在于 SA 中。

(2)用户请求访问某文件 f_x,若文件存在于 CA 中,则跳到步骤(4),若不存在于 CA 中,则顺序往下执行。

(3)系统通过 ARC 算法,换出那些存在于 CA 中但访问频率低的文件,并将 f_x 加入 CA 之中(详细的替换过程请见第 4.3 节)。

(4)用户成功访问文件 f_x,访问完成之后跳到步骤(2),继续进行下一轮的文件访问。

对于云存储文件系统的用户来说,主要有写入文件(即上传文件)、读取文件、修改和删除文件这几个操作。本文的主要关注点在于读和写操作,故对修改和删除操作的讨论略过。当接收到用户的操作请求之后,系统具体的响应过程如图 1 所示。

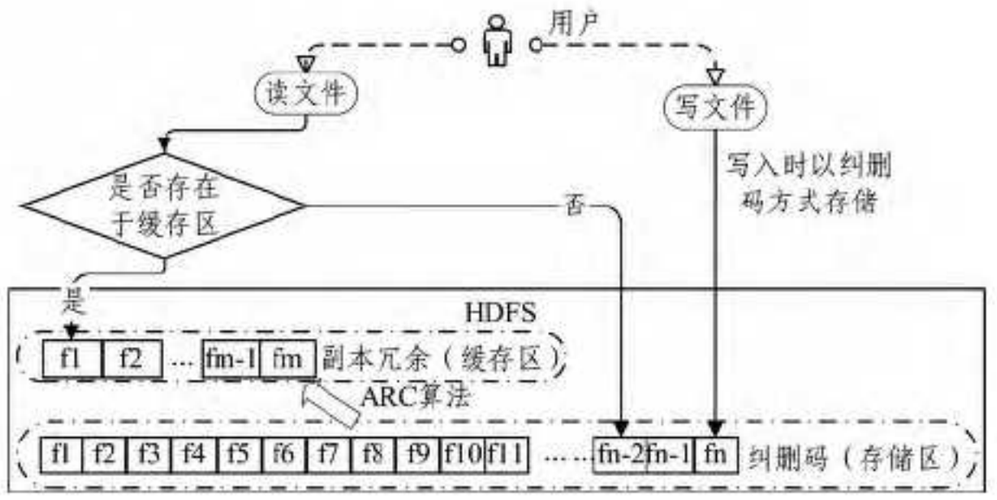

图 1 系统的响应示意图

4.2 工作机制

在目前所广泛使用的云存储文件系统中(比如 GFS、HDFS 等),系统一般采用的是副本冗余存储策略。采用副本冗余的存取方式,当存储规模较大时,空间的消耗是巨大的。纠删码能够很好地节约文件的存储空间,同时还能提高存储系统的容错性。但当文件损坏或丢失时,纠删码的恢复成本就比副本冗余高出许多。

ARCMFF 是一种将纠删码与副本冗余结合起来存储文件的容错机制,它能够更好地节约空间,具有容错性强、访问快、出错恢复快的特点。

如何决定文件是用副本冗余方式,还是纠删码方式存储的具体工作机制如下:

1)文件读和写

首先,当用户初次将某个文件写入文件系统时,为了节省存储空间,系统将以纠删码的方式将文件存入 HDFS 中。接下来,根据 HDFS 上的文件读取的频率及历史记录,系统会将那部分读取较为频繁的文件加入文件的缓存系统中去。

当用户需要读取一个文件时,若文件存在于缓存,用户就能直接从缓存中读取所需要的文件。若文件不存在于缓存中,那么系统将从 HDFS 中读取出文件的纠删码。

2)缓存系统

对于文件默认采用的纠删码存储方式,其读取速度略慢于副本冗余。为了达到快速读取文件的目的,系统会将频繁读取的文件加入缓存系统中,也即文件用副本冗余方式存储。缓存系统中的文件(即以副本冗余方式存储的文件)具有读取速度快,但占用存储空间较大的特点。

由于系统的缓存空间有限,对于那些位于缓存中,但又不常被读取的文件,系统会将其从缓存系统中删除。而那些未加入缓存系统中,但又访问频繁的文件,系统会将其加入缓存中。

3)容错机制

在文件的实际存储时,当某一文件发生错误,导致文件部分失效时,系统将通过一定的容错机制来恢复文件。

若文件存在于缓存中,即文件是以副本冗余方式存储,那么系统将从其任一未失效副本中,将文件恢复;若文件不存在于缓存中,即文件是以纠删码方式存储,系统将从文件的纠删码中将其恢复。具体来说,若将某文件分为 k 个数据块,通过

一定的编码计算得到 m 块冗余元素。那么对于这 $k+m$ 个数据块，系统只要从其中找出任意 k 块未出错的数据块，就可以通过对应的重构算法恢复出原来的 k 块数据。

对于副本冗余来说，只需要简单地从一个正常的副本中拷贝一份文件即可，具有出错恢复速度快的特点。但对纠删码来说，当一个数据块出错时，系统需要读取出多个数据块，并通过一定算法才能将原始数据块恢复，故纠删码的出错恢复性能不是很好。

4.3 详细设计

经过前面的介绍可知，ARCMFF 在云储存文件系统中（文中以 HDFS 为例），基于 ARC 算法动态地调整系统中文件的副本冗余或纠删码方式存储，从而节省了可观的文件存储空间。

从图 1 中可以看到，ARC 算法可以从 SA 中挑选出部分频繁访问的文件存入 CA 中。下面为 ARC 算法的具体调度过程（代码中 ADAPTATION: 标记处为 ARCMFF 的自适应调整函数）。

设文件的访问序列为 $f_1, f_2, f_3, \cdots, f_t$，算法的伪代码实现如下（需注意，文件 f_t 若存在于 T_1 或者 T_2 中，那么文件即是加入 CA 中，若 f_t 从 T_1 或者 T_2 中移出，那么文件即从 CA 中移除）：

INPUT: The file request stream: $f_1, f_2, \cdots, f_t, \cdots$

INITIALIIZATION: Set p=0 and set the LRU lists T_1, B_1, T_2 and B_2 to empty

For every $t \geqslant 1$ and any f_t, one and only one of the following four cases must occur.

Case I: f_t is in T_1 or T_2. A cache hit has occurred

Move f_t to MRU position in T_2 (f_t is already in the CA)

Case II: f_t is in B_1. A cache miss has occurred

ADAPTATION: Update $p=\min\{p+\delta_1, c\}$, where

$$\delta_1=\begin{cases}1, & \text{if } |B_1| \geqslant |B_2| \\ \dfrac{|B_2|}{|B_1|}, & \text{otherwise}\end{cases}$$

REPLACE(f_t, p), Move f_t from B_1 to the MRU position in T_2 (also fetch f_t to the CA)

Case III: f_t is in B_2. A cache miss has occurred

ADAPTATION: Update $p=\max\{p-\delta_2, 0\}$, where

$$\delta_2=\begin{cases}1, & \text{if } |B_2| \geqslant |B_1| \\ \dfrac{|B_1|}{|B_2|}, & \text{otherwise}\end{cases}$$

REPLACE(f_t, p), Move f_t from B_2 to the MRU position in T_2 (also fetch f_t to the CA)

Case IV: f_t is not in $T_1 \cup B_1 \cup T_2 \cup B_2$. A cache miss has occurred

Case A: $L_1=T_1 \cup B_1$ has exactly c pages.

If($|T_1|<c$)

Delete LRU page in B_1. REPLACE(f_t, p).

else

Here B_1 is empty. Delete LRU page in T_1 (also remove it from the CA)

endif

Case B: $L_1=T_1 \cup B_1$ has less than c pages.

If($|T_1|+|T_2|+|B_1|+|B_2| \geqslant c$)

Delete LRU page in B_2, if($|T_1|+|T_2|+|B_1|+|B_2|=2c$)

REPLACE(f_t, p)

endif

Finally, fetch f_t to the CA and move it to MRU position in T_1.

Subroutine REPLACE(f_t, p)

If((|T_1| is not empty) and((|T_1| exceeds the target p) or(f_t is in B_2 and |T_1|=p)))

Delete the LRU page in T_1 (also remove it from the CA), and move it to MRU position in B_1

else

Delete the LRU page in T_2 (also remove it from the CA), and move it to MRU position in B_2

endif

通过以上过程，系统就能动态地将副本冗余和纠删码方式结合起来存储文件，节省了大量的存储空间。而通过 ARC 算法的调度，系统能够达到很高的缓存命中率，从而拥有较好的文件访问性能。

5 实验及结果分析

为了验证本文所提出的机制的可行性，我们在 Hadoop 平台上进行了多次实验。

5.1 实验环境

实验集群由 1 台交换机，6 台主机组成。其中 1 台主机作为集群的 NameNode 和 SecondaryNamenode，4 台主机作为集群的 DataNode，另有 1 台电脑专门作为客户端做下载测试用。

实验的 6 台主机硬件配置参数均相同，CPU 为双核 2.3 GHz，硬盘为 160G，内存为 DDRII，1G。

为了方便和保证实验结果的可靠性，本实验中，所有电脑的操作系统均为 Ubuntu 12.04 LTS；使用的 Hadoop 版本均为 1.2.1；纠删码采用的是开源的 C 语言版 Jerasure，为了实现对 Hadoop 的兼容，已用 Java 语言重写；JDK 的版本为 JDK 1.6。

5.2 实验过程

本实验是基于 Hadoop 进行测试的，在集群外的一台测试机上分别对 4 种（10M、50M、100M、200M）不同大小级别的文件进行操作，测试并分析其性能。

以 10M 大小级别的文件为例，我们在 HDFS 中上传了 100 个 10M 大小的文件 10M-0，10M-1，10M-2……10M-99，缓存空间设为 10 个文件的大小。在测试机，从这 100 个文件中随机读取 500 次文件。相关研究表明，互联网上的文件访问是符合 Pareto 分布的，因此本实验随机读取文件所采用的随机函数是 Pareto[20] 函数，实验结果如下。

5.3 采用与未采用该机制的性能对比

首先，本实验分别针对采用 ARCMFF、默认机制（即单一副本冗余机制）和单一纠删码机制容错的 HDFS 进行写入时间、读取时间以及存储占用空间的测试。

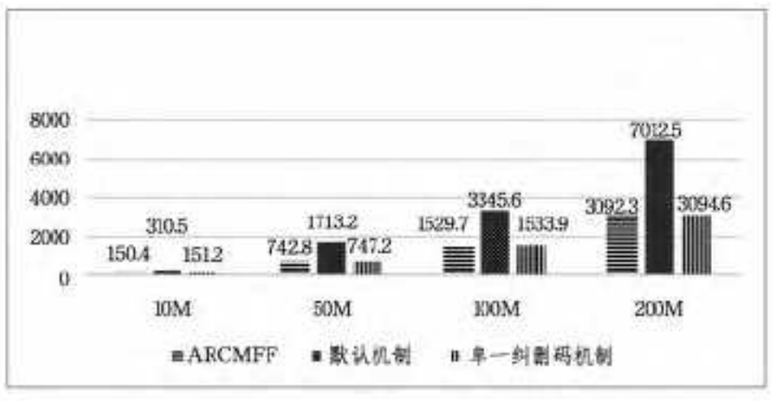

图 2 文件写入时间(s)对比

从图 2 中可以看出，采用该机制和单一纠删码机制的写入时间差别很小。在该机制下，文件上传时，文件默认就是以

纠删码方式进行存储的，故两者的写入时间基本相同。同采用默认机制相比，采用该机制后至少节省了50%以上的写入时间。

从图3中可以看到，相比默认机制，采用该机制后，文件的读取时间略有增加，但增加幅度都在10%以内，且当文件大小级别增大时，文件读取时间的增加幅度在2%以内。

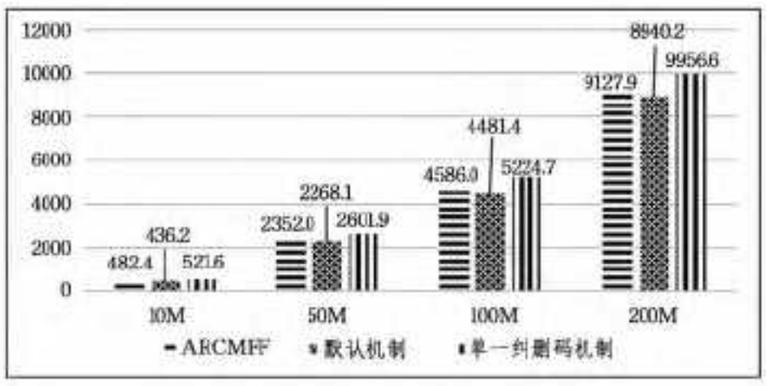

图3 文件读取时间(s)对比

图4表明，1单位大小的文件若采用该机制存储，存储空间接近于1.6单位大小，而采用默认机制(副本冗余)存储的HDFS将消耗接近于4单位大小的空间，较之节省了将近60%的存储空间。

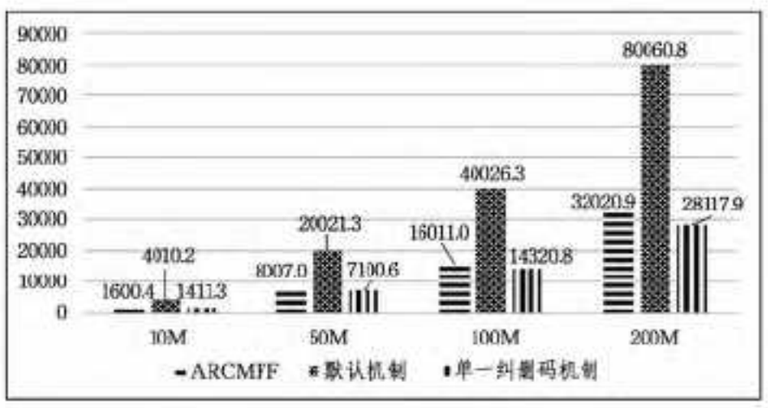

图4 文件占用空间大小(M)对比

5.4 不同算法之间的性能对比

本文前面的实验是将ARC缓存替换算法应用到云存储文件系统中，动态调整副本冗余和纠删码来进行实验。除此之外，本文还将缓存调度过程中起关键作用的ARC算法，替换成同类的缓存替换算法(FRC、FIFO、LFU、LRU、MQ、TwoQ等)，并进行了性能测试，结果如图5所示。

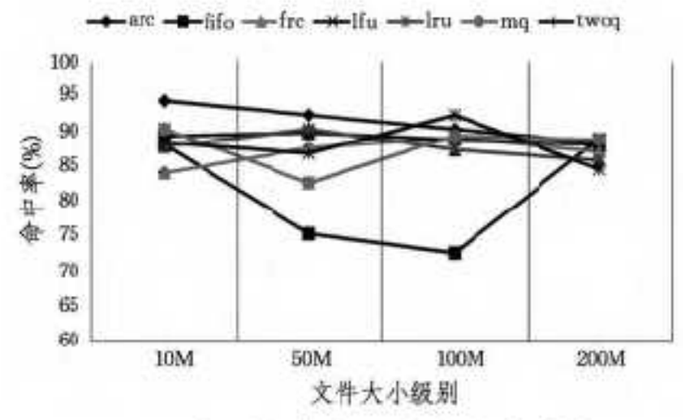

图5 各缓存算法平均命中率对比

在替换了缓存调度算法后，它们的性能各不相同。很明显，由于更换的仅仅是算法的类型，且在每个小组实验中，每个文件都是等大小的，故不同算法的文件写入、读取时间和文件存储占用空间的大小都是相同的。对于不同的缓存调度算法来说，它们缓存命中率却是不同的。缓存命中率也是衡量该机制性能的一个重要指标。缓存命中率越高，那么在缓存系统中文件的变化也就越小，从而在替换缓存时所进行的写操作也就越少，读取时间也越少。总体来说，缓存命中率的高低对HDFS的影响较大。从图5中可以看出，采用不同算法，ARC的平均命中率最高，FIFO的平均命中率最低，但都在70%以上，但在不同的应用场景下，不同算法的命中率好坏可能会发生变化。比如在一个符合程序局部性访问原理的文件访问模型中，采用LRU算法的性能最佳。而在一个基于访问频率的文件访问模型中，采用LFU算法的性能最佳。在实际应用场景中，文件访问并非总是符合某个特定的访问模型，而是经常发生变化的。ARC算法的优势在于，它能够根据应用场景的历史记录，自动地进行调整，以适应变化的文件访问模型，从而获得很高的缓存命中率。

5.5 结论

从实验结果可以看出，与采用副本冗余机制的HDFS相比，采用该机制后，文件读取时间虽然略有增加，但幅度很小。但却节省了近60%的存储空间，写入时间也减少了50%以上。通过ARCMFF，可用接近于单一纠删码的存储空间及文件写入时间获得接近于单一副本冗余机制的文件读取性能。

结束语 容错机制是分布式文件系统保证文件正常访问过程中尤为关键的一部分。本文在已有的关于副本冗余和纠删码研究的基础上，提出了一种将两者结合起来的容错机制ARCMFF。在一个副本冗余和纠删码共存的文件系统中，它创新地将纠删码存储的文件等效为内存中的页面，副本冗余存储的文件等效为Cache中的页面。通过借鉴内存和Cache之间的缓存替换算法ARC，从缓存调度的角度来决定文件的存储方式。同时ARC缓存调度算法能在文件访问模型发生变化时，自适应地调整缓存调度策略。即使在访问模型经常发生变化的实际应用场景中，ARCMFF也能拥有很高的缓存命中率，因此采用了该机制的分布式文件系统能够拥有很好的文件存储和文件访问性能。实验表明，采用ARCMFF后，分布式文件系统能够节省大量的存储空间且可以减少文件的写入时间。在后续的研究工作中，将测试在出现存储故障时，系统的容错及出错恢复性能等。

参考文献

[1] 郭全中，郭凤娟. 大数据时代下的媒体机遇[OL]. http://media.people.com.cn/n/2014/0304/c192370-24525582.html

[2] Pinheiro E, Weber W D, Barroso L A. Failure trends in a large disk drive population[C]// Proc of the 5th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2007: 17-28

[3] Schroeder B, Gibson G A. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? [C]// Proc of the 5th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2007: 1-16

[4] Bairavasundaram L N, Goodson G R, Pasupathy S, et al. An analysis of latent sector errors in disk drives[C]// Proc of 2007 ACM SIGMETRICS IntConf on Measurement and Modeling of Computer Systems. New York: ACM, 2007: 289-300

[5] Satyanarayanan M, Howard J H, Nichols D A, et al. The ITC distributed file system: principles and design[M]. ACM, 1985

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

[7] Borthakur D. The hadoop distributed file system: Architecture and design[J]. Hadoop Project Website, 2007, 11: 21

[8] Palankar M R, Iamnitchi A, Ripeanu M, et al. Amazon S3 for science grids: a viable solution? [C]// Proceedings of the 2008 international workshop on Data-aware distributed computing. ACM, 2008: 55-64

(下转第340页)

响。刘海坤等[8]结合理论分析与指数加权移动平均方法，建模单虚拟机在线迁移，量化分析网络速率和虚拟机大小等因素对迁移性能的影响。但是上述模型均是基于传统的预拷贝策略，而不是基于 qemu 中的预拷贝策略。

在页面压缩方面，金海等[6]提出一种基于内存页面特征的自适应压缩算法压缩内存数据，降低数据传输量，缩短在线迁移的时间。Gupta 等[5]采用基于哈希的指纹技术(hash-based fignerprints)计算相似页面，并使用 Xdelta 计算相似页面之间的差异，实现 delta 页面传输。Svard 等[10]使用 2 路组相联高速缓存技术(2-way set associative cache)缓存已发送页面，然后使用异或操作计算待传页面与已传页面之间的差异，实现 delta 页面传输。张翔等[1]提出一种基于位图和异或压缩的方法计算磁盘文件的变化，并采用流水线技术同步文件传输，提高在线迁移的性能。

在页面传输方面，罗英伟等[9]提出一种基于块位图(block-bitmap)的方法记录内存数据块是否被改写，以此计算需要传输的页面，减少数据重传。陈阳等[2]结合主动推送和按需复制两种传输机制，提出一种混合内存复制方法，实现脏页面的快速复制，减少数据传输量。胡亮等[7]也提出一种类似的混合内存复制方法，但是他们将 delta 页面压缩机制融入到方法中，进一步减少数据传输量。

结束语　已有的动态迁移模型均针对传统的预拷贝迁移策略。开源虚拟化平台 qemu-kvm 中的动态迁移策略与之存在一些差异，这导致已有的动态迁移模型无法有效应用于 qemu-kvm 环境下的动态迁移性能预测。为此，本文提出一种基于 qemu-kvm 的动态迁移性能模型，着重分析影响迁移性能的关键因素，并针对性地进行实验测试，通过比较实验数据与基于模型推算的理论数据之间的偏差，验证模型的有效性与精确性。实验结果表明，本文提出的性能评估模型在估算迁移时间与数据传输总量方面的精确性在 95% 以上。

参 考 文 献

[1] 张翔，霍志刚，马捷，等. 虚拟机快速全系统在线迁移[J]. 计算机研究与发展，2012，49(3)：661-668

[2] 陈阳，怀进鹏，胡春明. 基于内存混合复制方式的虚拟机在线迁移机制[J]. 计算机学报，2011，34(12)：2278-2291

[3] Akoush S，Sohan R，Rice A，et al. Predicting the performance of virtual machine migration[C]// 2010 IEEE International Symposium on Modeling，Analysis & Simulation of Computer and Telecommunication Systems(MASCOTS). 2010：37-46

[4] Aldhalaan A，Menascé D A. Analytic Performance Modeling and Optimization of Live VM Migration[C]// Computer Performance Engineering. Springer，2013：28-42

[5] Gupta D，Lee S，Vrable M，et al. Difference engine：Harnessing memory redundancy in virtual machines[J]. Communications of the ACM，2010，53(10)：85-93

[6] Jin Hai，Li Deng，Wu Song，et al. Live virtual machine migration with adaptive memory compression[C]// IEEE International Conference on Cluster Computing and Workshops，2009(CLUSTER'09). 2009：1-10

[7] Hu Liang，Zhao Gao，Xu Gao-chao，et al. HMDC：Live Virtual Machine Migration Based on Hybrid Memory Copy and Delta Compression[J]. Applied Mathematics & Information Sciences，2013，7：639-646

[8] Liu Hai-kun，Jin Hai，Xu Cheng-zhong，et al. Performance and energy modeling for live migration of virtual machines[J]. Cluster computing，2013，16(2)：249-264

[9] Luo Ying-wei，Zhang Bin-bin，Wang Xiao-lin，et al. Live and incremental whole-system migration of virtual machines using block-bitmap[C]// 2008 IEEE International Conference on Cluster Computing. 2008：99-106

[10] Svärd P，Hudzia B，Tordsson J，et al. Evaluation of delta compression techniques for efficient live migration of large virtual machines[J]. ACM Sigplan Notices，2011，46(7)：111-120

[11] Wu Yang-yang，Zhao Ming. Performance modeling of virtual machine live migration[C]// 2011 IEEE International Conference on Cloud Computing. 2011：492-499

[12] Zhu Chang-peng，Zhao Yin-liang，Bo Han，et al. Runtime support for type-safe and context-based behavior adaptation[J]. Frontiers of Computer Science，2014，8(1)：17-32

(上接第 336 页)

[9] Chu Yu. 淘宝 TFS 的 wiki[OL]. http://code.taobao.org/p/tfs/wiki/index/

[10] McAuley A J. Reliable broadband communication using a burst erasure correcting code[J]. ACM SIGCOMM Computer Communication Review，1990，20(4)：297-306

[11] Weatherspoon H，Kubiatowicz J D. Erasure coding vs. replication：A quantitative comparison[M]// Peer-to-Peer Systems. Springer Berlin Heidelberg，2002：328-337

[12] Wu L，Liu B，Lin W. A Dynamic Data Fault-Tolerance Mechanism for Cloud Storage[C]// 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies(EIDWT). IEEE，2013：95-99

[13] 林伟伟. 一种改进的 Hadoop 数据放置策略[J]. 华南理工大学学报：自然科学版，2012，40(1)：152-158

[14] 利业鞾，林伟伟. 一种 Hadoop 数据复制优化方法[J]. 计算机工程与应用，2012，48(21)：58-61

[15] 林伟伟，刘波. 基于动态带宽分配的 Hadoop 数据负载均衡方法[J]. 华南理工大学学报：自然科学版，2012，40(9)：42-47

[16] 林伟伟，贺品嘉，刘波. 云存储系统的能耗优化节点管理方法[J]. 华南理工大学学报：自然科学版，2014，42(1)：104-110

[17] Megiddo N，Modha D S. ARC：A Self-Tuning，Low Overhead Replacement Cache[C]// FAST. 2003，3：115-130

[18] 罗象宏，舒继武. 存储系统中的纠删码研究综述[J]. 计算机研究与发展，2012，49(1)：1-11

[19] Lin W K，Chiu D M，Lee Y B. Erasure Code Replication Revisited[C]// Peer-to-Peer Computing. 2004：90-97

[20] 康殿统，王文娟，杨雯. 关于 Pareto 分布的一个综合研究[J]. 河西学院学报，2008，24(2)：1-5