doi:10.3772/j.issn.1002-0470.2013.11.001

基于分级编码的高可靠存储策略①

冯清青②[*,***] 孟 丹[**] 韩冀中③[**]

([*]中国科学院计算技术研究所计算机应用研究中心 北京 100190)
([**]中国科学院信息工程研究所 北京 100093)
([***]中国科学院大学 北京 100049)

摘 要 研究了适应当前大数据时代的数据可靠性存储,针对已有存储策略难以同时满足高可靠性存储和高空间利用的需求的问题,提出了一种面向大数据的高可靠低冗余分级编码存储策略。该策略考虑到数据因类型不同、生命周期不同而重要程度有别的特性,可为不同类型数据分别设定容错级别;将不同冗余度的容错编码方式在一套统一存储架构中实现,用一组简单参数设置为数据选择恰当的容错级别编码存储;通过动态降低历史数据的冗余度进一步减少存储空间开销。实验验证了其有效性。对重要小文件采用高容错级别的编码分片存储,能在系统95%存储节点失效的情况下,根据编码后的部分数据分片快速修复所有数据;对普通文件采用适当放松的容错编码级别,在保证数据快速、无损修复的前提下比传统3副本策略节省1.5倍的存储空间。

关键词 大数据,存储,可靠性,容错,低冗余,分级,编码

0 引 言

信息技术的发展,使得全球数据量的增长及存储数据的规模达到了惊人的程度。这已由全球大互联网企业的运营数据所证明。Facebook 每天新增压缩数据12TB,每天扫描压缩数据800TB;Hadoop 分布式文件系统(hadoop distributed file system,HDFS)[1]存储文件量达65 million;Yahoo! 每天处理的数据规模达 PB 级,Google 处理的数据规模约为24PB/天;EMC 新闻公报指出如今全球数据增长速度已经远超过摩尔定律[2]。这表明,当今信息社会已进入大数据时代;庞大的数据规模和迅猛的数据增长速度给现代数据中心的数据存储可靠性研究带来了新的问题。

传统分布式文件系统普遍采用多数据副本容错策略。如 Google 文件系统(Googlefile system,GFS)[3],HDFS,Amazon 存储服务(Amazonsimple storage service,Amazon S3)[4]均采用多副本策略,系统默认数据副本数为3。多副本策略较好满足可靠性的同时无可避免地引入了巨大的空间开销及存储代价。除多副本容错策略外,数据编码是另一类重要的存储可靠性研究方法。对比多副本策略,编码容错技术可以用更小的存储空间开销提供同样的可靠性保证[5]。但是,现有编码容错技术普遍采用全局统一的编码策略,如 RAID5[6]、RAID6[7,8]编码,分别仅能解决一至两个数据块失效问题,难以满足不同存储数据的多种可靠性需求。本文针对这一问题进行了研究,提出了一种全新的分级编码存储策略,以解决多数据副本容错策略存储空间开销过大,全局统一的编码策略难以满足不同储存数据的可靠性需求的问题。

1 相关研究

1.1 数据副本策略

当前流行的 Hadoop HDFS 即采用多份数据副本来保证存储可靠性。通常,HDFS 默认保留3份

① 863 计划(2012AA01100,2012AA01A401),国家自然科学基金(61070028,61003063)和中国科学院先导专项(XDA06030200)资助项目。
② 女,1982 年生,博士生;研究方向:计算机系统结构,大数据存储;E-mail:fengqingqing@ncic.ac.cn
③ 通讯作者,E-mail:hanjizhong@iie.ac.cn
(收稿日期:2012-12-05)

数据副本,副本放置策略是在本地磁盘保留一份数据,在远程机架不同节点上再保留两份数据副本。由于存放同一数据副本的不同机架上两个节点同时宕机的可能性很小,这种多副本策略能较好保证分布式系统中数据的可靠性。此外,GFS 以及 Amazon S3,微软的 Windows Azure 存储系统(Windows Azure storage,WAS)[9],IBM 的 Spinnaker[10],均采用默认 3 副本的容错策略。为进一步验证存储数据的正确性,Amazon S3 在 3 副本基础上增加了 checksum 编码;IBM 的 Toshio's[11],除采用 3 副本之外,还增加了任务失败检测机制。除默认 3 副本外,以上系统还可设置全局统一的多份数据副本。此外,Amazon Dynamo[12] 则采用了一种基于洪泛协议的广播复制(epidemic replication)多副本容错策略。

全局统一的多副本策略,容易造成系统存储空间利用率不高。为进一步合理利用存储空间,增强副本配置的灵活性,Yahoo! PNUTS[13] 为数据及元数据分别设置不同的副本份数;Skute[14] 则根据应用特征不同分别配置不同副本份数。这种根据数据类型不同,或者根据应用特征不同而分别设置不同副本份数的多副本策略,减少了单一化副本配置带来的存储空间浪费。

1.2 数据编码策略

除多副本策略之外,数据编码是另一种被广泛使用的存储可靠性方法。最常用的容错编码技术为里德 - 所罗门(Reed-Solomon,RS)编码[15]。在 RS 编码中,原符号被视为有限域内多项式 $p(x)$ 的系数,通过对 $p(x)$ 过采样,从原始的 k 个符号中创建出 n 个编码符号。RS 编码通过引入 $n-k$ 个冗余符号信息达到容错的目的,用 RS 编码技术可以检测出 $t=(n-k)$ 个符号位错误,并纠正 $t/2$ 个符号位错误。纠删码(erasure Codes)[16] 将 RS 编码引入磁盘容错技术,将存储对象分割为 m 块,通过增加冗余,使编码后的数据达到 $n(n>m)$ 块,纠删码的关键特性是能从任意 m 个分块中恢复出系统编码之后的全部 n 个分块,从而恢复原始数据。独立碾盘冗余阵列(redundant arry of independent,RAID)技术[17] 是纠删码的一个特例。其中 RAID5 通过加入一块冗余校验块,能检测两块数据文件错,并纠正一块数据文件错误;RAID6 加入两块冗余校验块,可以检测出 4 个文件分块错,并能最多修复两个数据块错误。当错误的数据分块数大于 2 时,RAID6 编码无法修复出受损数据块。

近年来,Gibson 等人的研究工作[18] 提出:结合 HDFS 多副本策略与 RAID 技术可适当减少多副本带来的空间开销,同时提供可靠的数据存储。研究分析表明,该思想能较好解决由多副本引起的长期磁盘空间开销过大问题,也能利用 RAID 技术提供良好的数据可靠性。

本文结合不同数据类型以及数据在生命周期不同阶段重要程度不同的特性,提出了一种新的分布式分级存储架构。它使用统一的数据管理平台,由一组简单的 (n,k) 参数设定,即可实现为数据设定不同容错级别的编码存储方式。根据设定好的参数,用 (n,k) 编码算法将原始数据编码后分为 n 片分散存储于分布式盘阵中;从分布式盘阵获取任意 $k(k<n)$ 个数据分片,用 (n,k) 解码算法,即可还原出原始数据。通过调整 (n,k) 值可以实现存储系统整体的低冗余与高可靠特性。实验中,根据存储数据类型及文件大小分类,为重要的小文件设定高容错级别——$(n:k)=(18:1)$;为普通文件设定中等容错级别——$(n:k)=(2:1)$;为不太重要的大文件设定较低容错级别——$(n:k)=(1.5:1)$。本文的主要贡献在于:

(1)提出了一种新的分级编码存储架构,实现了海量数据的低冗余高可靠存储,很好地解决了存储空间利用率与数据可靠性之间的矛盾;

(2)用一套统一架构,将复杂的分级编码、容错级别控制问题简化为对一组 (n,k) 参数的设置与调整;

(3)大文件的空间开销比 HDFS 3 副本策略减少了 50%;

(4)重要小文件的容错能力比 3 副本策略提高了 $n/3$ 倍(n 为有效存储节点个数,$n\leq 3$ 时该值为 1)。

2 挑战与解决方法

本文工作致力于解决大数据时代多副本或单一编码策略无法同时满足存储系统高可靠与低冗余的问题。这项工作主要面临以下重大挑战:如何解决存储系统空间利用率与数据可靠性之间的矛盾?

经研究分析,不同类型的数据,或数据在生命周期的不同阶段,其重要程度有别。考虑到数据重要程度不同的特性,本文提出并设计了一种针对不同数据类型分级编码的存储策略。

为使复杂的分级管理过程可以在一套统一架构下变得简单可行,本文采用了如下基于 (n,k) 编码

的存储策略,工作过程如下:

(1)采用(n,k)编码算法将数据编码后分为n片分散存储于分布式盘阵;

(2)当数据丢失或错误时,采用(n,k)解码算法从$k(k<n)$个数据分片还原数据。

2.1 高可靠存储

(n,k)算法实现数据编码、解码的原理如图1所示。

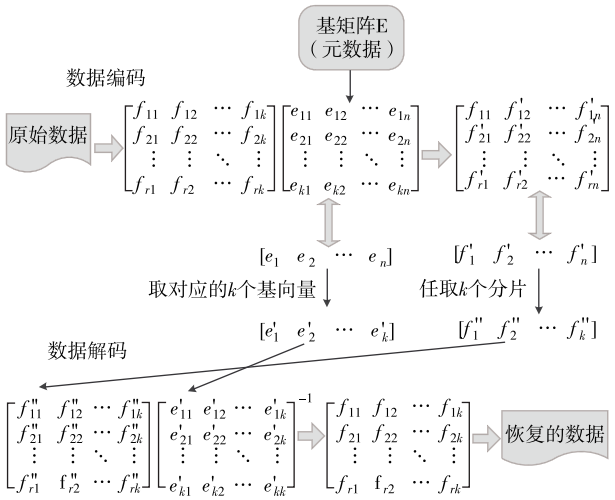

图1 数据编码解码过程

(a)在有限域内生成基矩阵E。其中E为$k\times n$矩阵,且满足矩阵E的任意$k(k<n)$个k维向量线性无关(n、k的值等于设定的参数值(n,k))。

(b)数据编码分片过程如式

$$\begin{aligned}\boldsymbol{F}\cdot\boldsymbol{E}&=\begin{bmatrix}f_1\\f_2\\\vdots\\f_r\end{bmatrix}\cdot[e_1\quad e_2\quad\cdots\quad e_n]\\&=\begin{bmatrix}f_1\\f_2\\\vdots\\f_r\end{bmatrix}\cdot\begin{bmatrix}e_{11}&e_{12}&\cdots&e_{1n}\\e_{21}&e_{22}&\cdots&e_{2n}\\\vdots&\vdots&\cdots&\vdots\\e_{k1}&e_{k2}&\cdots&e_{kn}\end{bmatrix}\\&=[f'_1\quad f'_2\quad\cdots\quad r'_n]\\&=\boldsymbol{F}'\end{aligned}\tag{1}$$

所示。将原始数据文件转换为矩阵$\boldsymbol{F}$,用基矩阵$\boldsymbol{E}$与原始数据$\boldsymbol{F}$做矩阵乘运算,得到新矩阵$\boldsymbol{F}'$,$\boldsymbol{F}'$即为编码后的数据文件,$\boldsymbol{F}'$的每一个向量可视为一个编码后的文件分片。

(c)数据解码还原过程如式

$$\begin{aligned}\boldsymbol{F}''\cdot\boldsymbol{E}'^{-1}&=\begin{bmatrix}f'_1\\f'_2\\\cdots\\f'_r\end{bmatrix}\cdot[e'_1\quad e'_2\quad\cdots\quad e'_k]^{-1}\\&=\begin{bmatrix}f'_1\\f'_2\\\cdots\\f'_r\end{bmatrix}\cdot\begin{bmatrix}e'_{11}&e'_{12}&\cdots&e'_{1k}\\e'_{21}&e'_{22}&\cdots&e'_{2k}\\\vdots&\vdots&\cdots&\vdots\\e'_{k1}&e'_{k2}&\cdots&e'_{kk}\end{bmatrix}\\&=\boldsymbol{F}\end{aligned}\tag{2}$$

所示。当取得n片中的任意k个数据分片时,用这k个数据分片组成一个$k\times k$矩阵$\boldsymbol{F}''$;从基矩阵$\boldsymbol{E}$取这k个分片对应的基,组成一个新的还原基矩阵$\boldsymbol{E}'$;用$\boldsymbol{F}''$和$\boldsymbol{E}'^{-1}$做矩阵乘运算还原原始数据文件$\boldsymbol{F}$。

为保证运算过程无精度损失,计算均在有限域内完成。文件编码分片和解码还原过程如图1所示。

2.2 低空间开销

由图1可见,编码之后的数据总量$\boldsymbol{F}'$是编码前的数据量$\boldsymbol{F}$的n/k倍。为节省存储空间,可选择n个数据分片中的任意$m(k<m<n)$个分片存储。此时,系统的空间开销即为原来的m/k倍,空间冗余量为$m/k-1$。

编码完成后,用哈希算法将数据分片均匀分散存储于集群中各节点。解码时,只需取到n(或m)个分片中的任意k片即可还原原始数据。

3 设计与实现

分级存储原型系统架构如图2所示。系统主要包含容错组件和存储组件两大功能模块。容错组件为本文工作的一个重要贡献,后文将详细介绍该部件设计与原理;另一个重要工作建立于对HadoopYarn[19] HDFS文件系统的改进。

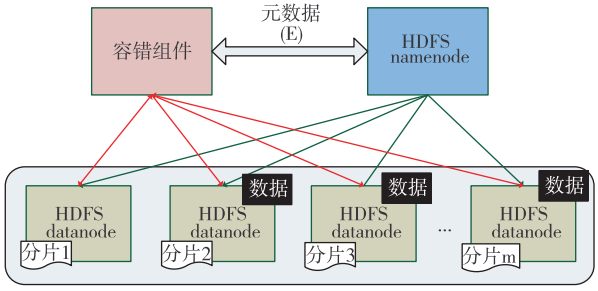

图2 分级存储系统架构

3.1 容错组件

容错组件用于设定和修改数据容错级别,根据所选容错级别执行数据编码分片,分散存储各分片;执行数据解码还原。容错组件内部结构如图 3 所示。

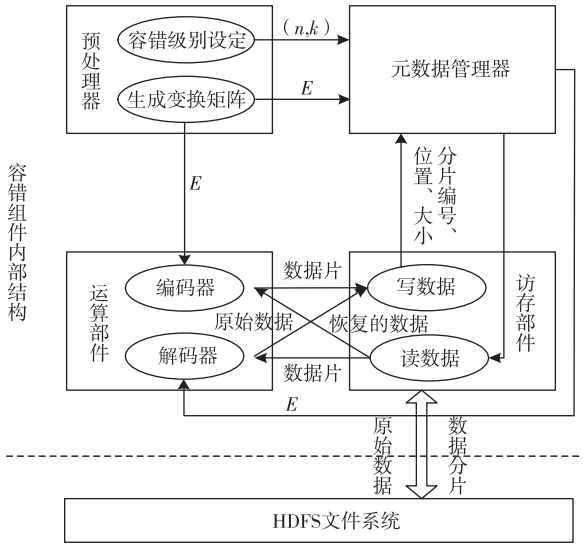

图 3 容错组件结构图

容错组件主要由预处理器、元数据管理器、运算部件和访存部件组成。各部分功能如下:

(a)预处理器包含容错级别设定和基矩阵生成模块。可根据数据类型选择并设定容错级别,实验中为重要小文件设置的最高容错级别为 $(n:k)=(18:1)$;为一般数据设置普通容错级别,$(n:k)=(2:1)$;为不太重要的大文件设置较低容错级别,$(n:k)=(1.5:1)$。基矩阵生成模块,根据容错级别生成变换基矩阵 $\boldsymbol{E}$。

(b)运算部件由编码器和解码器两个模块组成。编码器接收预处理器生成的变换基矩阵 $\boldsymbol{E}$,通过访存部件从 HDFS 文件系统取得待编码的原始数据,执行有限域内的矩阵乘数据编码运算。

解码器将取得的数据分片重组为还原矩阵;取基矩 $\boldsymbol{E}$ 中对应的基向量生成还原基矩阵 $\boldsymbol{E'}^{-1}$;执行数据解码运算。

(c)访存部件负责容错组件与 HDFS 文件系统间的读写交互,包括写数据模块和读数据模块。写数据模块含两项功能:第一,向 HDFS 写入编码后的数据片,将各数据分片的编号、所在节点编号、分片大小等元数据信息汇报给元数据管理器;第二,为用户提供恢复之后的数据文件。

读数据模块功能如下:第一,从 HDFS 文件系统读原始数据;第二,获取元数据信息,从 HDFS 系统读取编码之后的数据分片。

(d)元数据管理器,不同 HDFS Namenode。管理基矩阵 $\boldsymbol{E}$,数据分片的编号、位置、分片大小等信息。

3.2 存储组件

存储组件设计于 Hadoop HDFS 文件系统基础之上。为实现分级存储系统低冗余的目标,在 HDFS 基础上做了如下改进:

(1)调整 HDFS 分块策略,使 Datanode 既可存放原始数据文件,也可存编码后的数据分片。

(2)为原始数据添加一个状态标签,作为新增元数据信息存放于 Namenode。所有新读入数据被标记为“hot”(热数据);在对该数据的读写访问、编码分片以及存储过程完成后,将数据状态重置为“cold”(冷数据)。

(3)Namenode 通过心跳定期检测系统中原始数据的状态,当检测到数据状态为“cold”即为可删除数据,根据需要可随时删除“cold”数据以释放磁盘空间。

4 测试与实验

为分析分级存储策略在不同容错级别下的数据编码和错误恢复时间,检测分级存储系统 I/O 性能,本文进行了如下实验。实验基础平台为 Amazon ec2 Extra Large Instance(Standard Instances)节点上搭建的 Hadoop 集群。每个节点 15 GB 虚拟内存(实际 8GB 物理内存),8 个 EC2 Compute Units(4 virtual cores with 2 EC2 Compute Units each);基本软件环境:Ubuntu 12.04 LTS, hadoop-2.0.0-alpha, jdk-1.7.0_03。实验集群均为 1 个 master,多个 slave 结构。

4.1 分级数据编码、恢复时间

本组实验集群规模为 20 个 Amazon ec2 Extra Large Instance 节点,以 hadoop-2.0.0-alpha 搭建基础集群环境,用分级存储系统 NKFS 代替 HDFS 存储,检测不同容错级别下数据编码以及恢复时间。

测试中取 $n=18$,$k=12,9,3,2,1$。即 $(n:k)=(1.5:1)$、$(2:1)$、$(6:1)$、$(9:1)$、$(18:1)$ 五种容错级别,实验结果见图 4 至图 7。横坐标表示数据规模,纵坐标表示数据处理时间,时间单位为秒。

图 4 为 20 个节点集群下不同容错级别的数据

编码时间,由图可见随(n:k)比值越大,编码时间越长;当数据规模小时,随着数据量增大编码时间增加,当数据规模达到一定程度,编码时间趋近某一个上限值,该值与(n:k)的参数设定以及集群节点规模相关,集群规模越大,该值越小。

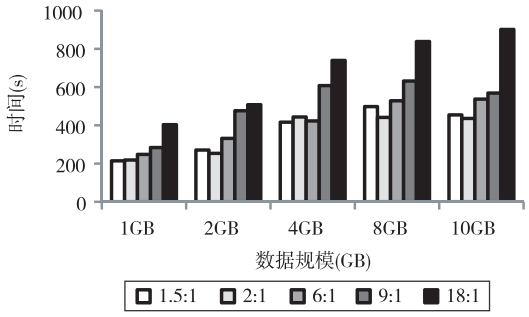

图 4　数据编码时间

图 5 代表不同容错级别下,数据恢复的时间。在(n:k) = (18:1)时,即使系统中约 95% 的存储节点失效,也可从剩余 5% 的节点中恢复出原始数据,且恢复 1GB 数据时间不超过 400s,恢复 10GB 数据也仅需 600 多 s。数据恢复时间随(n:k)比值增大而增长,随着数据规模增加而呈上升趋势,当数据规模增加到一定程度,恢复时间增长量越来越小。

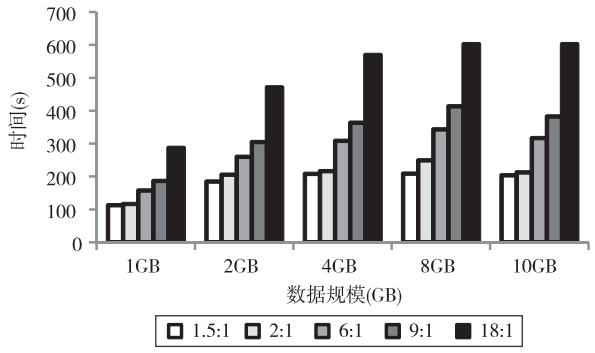

图 5　数据恢复时间

4.2　文件系统 I/O 性能

为测试分级存储系统下的 I/O 性能,我们使用 Hadoop 的 TestDFSIO 基准测试程序,对比测试 HDFS 和 NKFS(分级存储系统)文件读写性能。

图 6、图 7 分别表示两种存储系统执行 TestDFSIO 写文件、读文件操作的时间,时间单位为秒。其中 NKFS 的参数设置为(n,k) = (10,5)即(n:k) = (2:1)。测试集群规模为 11 个 Amazon ec2 Extra Large Instance 节点。

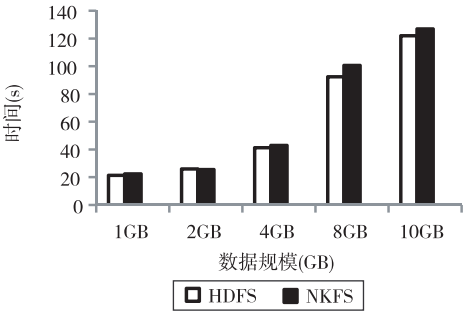

图 6　TestDFSIO write 时间

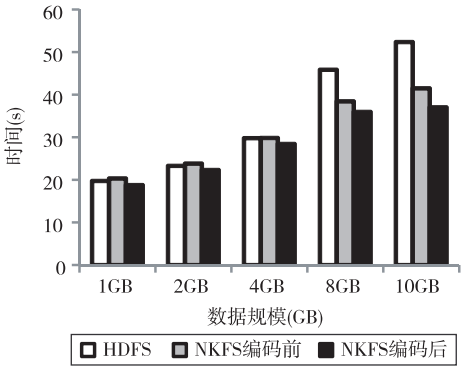

图 7　TestDFSIO read 时间

从图 6 可见分级存储策略基本不会影响写文件操作的性能。读文件测试中,由于 NKFS 编码之前采用和 HDFS 一样默认的 3 副本存储,读文件性能与 HDFS 3 副本相当;NKFS 编码后的读性能测试的是编码后存储系统中仅保留 k 个数据分片的存储方式,由图 7 可见编码后的数据存储方式不会影响系统读文件性能。

4.3　可靠性及代价分析

NKFS 分级存储系统与 HDFS、RAID 编码的可靠性及存储代价分析详见表 1。对比中选用 HDFS 默认的 3 副本;RAID 5 编码方式 3 +1 代表常用的 3 个数据块加 1 个校验块;RAID 6 编码方式 4 +2 代表 4 个数据块加 2 个校验块。NKFS 分级编码(18:1)、(2:1)、(1.5:1)代表该容错级别下(n:k)的比值。n 代表编码后的总数据分块数。

由表 1 可见,NKFS 比 HDFS 3 副本,RAID 5、RAID 6 编码具有更高容错能力。但是,NKFS(18:1)的编码方式在增强系统可靠性同时也带来了 18 倍的存储开销。这种分级存储策略能否有效节省存储空间? 根据不同数据类型,我们进行了如表 2 所示分析。

表 1 NKFS 分级编码与 HDFS 及 RAID 编码容错能力、存储成本对比

	编码方式	容错能力	存储成本
HDFS 3 副本	Null	2 份副本丢失	300%
RAID 5 编码	3 + 1	1 个数据块错	133.3%
RAID 6 编码	4 + 2	2 个数据块错	150%
NKFS 分级编码	(18:1)	95% × n 个数据块错	18 倍
	(2:1)	50% × n 个数据块错	200%
	(1.5:1)	33.3% × n 个数据块错	150%

表 2 NKFS 分级存储系统整体存储开销分析

容错级别	数据类型	数据规模	存储开销
高	重要文件（密码、关键字等）	B ~ kB	18 倍
普通	普通文本	kB ~ MB ~ GB	200%
较低	媒体文件（图片、音频、视频等）	MB ~ GB ~ TB	150%
系统整体开销	150% ~200%（主要由大文件决定）		

由于高容错级别编码的重要文件数据规模仅有几 B 至几 KB，系统整体存储开销主要由数据规模为 MB、GB 甚至 TB 的大量普通文件及媒体文件决定，通过为这些文件设置合理的容错级别，NKFS 的系统整体空间开销可接近 150%。

4.4 性能及存储优化

为达到性能、可靠性及存储空间利用率的进一步优化，本文采取了如下策略：

为降低编码带来的性能影响，将数据编码分片过程放在后台进程，与前台客户请求同步执行。同时，为了避免去副本带来的性能影响，一开始采用了与 HDFS 默认相同的 3 副本策略，待后台数据分片存储完成，可根据需要随时删除或还原这类数据。

为达到尽可能高的容错能力，将数据分片尽可能分散存储于集群各节点，数据分片在集群中分布越均匀，存储数据的可靠性越高。

随时间推移，部分历史数据的重要程度也可能降低。对这类数据，在删掉 3 份原始数据之后再删除 $r(0 \leqslant r \leqslant n-k)$ 个数据分片，只要保证系统中存储的分片个数大于等于 k，该数据就具有可恢复性。

通过删除 r 个分片，可以进一步把空间开销缩减至接近 100%。

5 结 论

本文的研究工作立足于解决大数据时代分布式存储系统数据高可靠和存储空间利用率不高的矛盾。鉴于目前常用的可靠存储策略——多数据副本存在存储空间开销过大，全局统一的容错编码方法难以满足不同存储对象的多样化可靠性需求，本文提出一种全新分级编码存储策略。它根据不同数据类型及同一数据在不同时间其重要程度有别的特性，结合 (n,k) 编码算法，为不同数据类型分别设定容错级别编码存储。通过合理的设置参数，可达到系统整体空间开销比 3 副本存储策略减少约 50% 前提下，重要小文件高度可靠。

实验结果表明，对重要文件本方法能从 95% 存储节点彻底失效的严重数据灾难下实现快速修复。

参考文献

[1] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, USA, 2010. 1-10

[2] Mcgaughey K. Worl' data more than doubling every two years—driving big data opportunity, new IT roles. http://www. emc. com/about/news/press/2011/20110628-01. htm, 2011

[3] Ghemawat S, Gobioff H, Leung S-T. The Google file system. In: Proceedings of the 19th ACM symposium on Operating systems principles, Bolton Landing, USA, 2003. 29-43

[4] Amazon C. Amazon simple storage service (Amazon S3). http://aws. amazon. com/s3/, 2012

[5] Rodrigues R, Liskov B. High Availability in DHTs: Erasure Coding vs. Replication. Springer Berlin Heidelberg: Springer Science + Business Media, 2005. 3640: 226-239

[6] Chen P M, Lee E K, Gibson G A, et al. RAID: high-per-

formance, reliable secondary storage. *ACM Computing Surveys (CSUR)*, 1994, 26(2): 145-185

[7] Stephenson D J. RAID architecture with two-drive fault tolerance. US patent: 6353895[P], 2002-3-5

[8] Jin C, Jiang H, Feng D, et al. P-Code: a new RAID-6 code with optimal properties. In: Proceedings of the 23rd international conference on Supercomputing, Yorktown Heights, USA, 2009. 360-369

[9] Calder B, Wang J, Ogus A, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In: Proceedings of the 23rd ACM Symposium on Operating Systems Principles, Cascais, Portugal, 2011. 143-157

[10] Rao J, Shekita E J, Tata S. Using Paxos to build a scalable, consistent, and highly available datastore. In: Proceedings of the VLDB Endowment, Seattle, USA, 2011. 243-254

[11] Suganuma T, Koseki A, Ishizaki K, et al. Distributed and fault-tolerant execution framework for transaction processing. In: Proceedings of the 4th Annual International Conference on Systems and Storage, Haifa, Israel, 2011. 1-12

[12] Decandia G, Hastorun D, Jampani M, et al. Dynamo: amazon's highly available key-value store. In: Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles, Stevenson, USA, 2007. 205-220

[13] Cooper B F, Ramakrishnan R, Srivastava U, et al. PNUTS: Yahoo! 's hosted data serving platform. *Proceedings of the VLDB Endowment*, 2008, 1(2): 1277-1288

[14] Bonvin N, Papaioannou T G, Aberer K. A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In: Proceedings of the 1st ACM Symposium on Cloud Computing, Indiana, USA, 2010. 205-216

[15] Reed I, Solomon G. Polynomial codes over certain finite Fields. *Society of Industrial and Applied Mathematics*, 1960, 8: 300-304

[16] Weatherspoon H, Kubiatowicz J. Erasure coding vs. replication: a quantitative comparison. In: Proceedings of IPTPS'01 Revised Papers from the 1st International Workshop on Peer-to-Peer Systems, Springer-Verlag London, UK, 2002. 328-338

[17] Plank J S. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software Practice and Experience*, 1997, 27(9): 995-1012

[18] Fan B, Tantisiriroj W, Xiao L, et al. DiskReduce: RAID for data-intensive scalable computing. In: Proceedings of the 4th Annual Workshop on Petascale Data Storage, Portland, USA, 2009. 6-10

[19] Foundation T a S. Apache Hadoop NextGen MapReduce (YARN). http://hadoop.apache.org/docs/r0.23.0/hadoop-yarn/hadoop-yarn-site/YARN.html, 2011

A high reliable storage architecture with hierarchical coding

FengQingqing[*,***], Meng Dan[**], HanJizhong[**]

([*] Institute of Computing Technology, Chinese Academy of Sciences, Beijing100190, China)

([**] Institute of Information Engineering, Chinese Academy of Sciences, Beijing100093, China)

([***] University of Chinese Academy of Sciences, Beijing100049, China)

Abstract

A study of high celiable data storage in the current big data age was conducted, and a novel hierarchical storage strategy for high reliable, low redundant storage of big data was proposed to solve the contradiction between the high reliability and the low storage utilization, facing the traditional storage strategies such as the multi-replication and the unified coding. To satisfy the diverse requirements of reliability for different storage objects, this strategy uses a unique architecture to provide variety of encoding methods for fault-tolerance. By setting the higher fault tolerance level for small text files and the lower fault tolerance level for large media files, the proposed strategy can bring the space overhead down from 200% to 50% compared with the triplication strategy. In addition, the small files will be recoverable even if 95% of storage node failures.

Key words: big data, storage, reliability, fault tolerance, low redundancy, hierarchical, coding