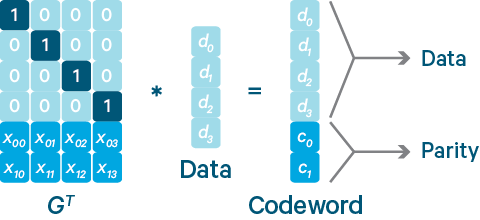
HDFS Erasure Code简介和使用

随着[大数据](http://lib.csdn.net/base/hadoop" \t "_blank" \o "Hadoop知识库)技术的发展，[HDFS](http://www.iteblog.com/archives/tag/hdfs" \t "_blank)作为[Hadoop](http://www.iteblog.com/archives/tag/hadoop" \t "_blank)的核心模块之一得到了广泛的应用。HDFS通过复本来实现数据的可用性，[HDFS](http://www.iteblog.com/archives/tag/hdfs" \t "_blank)中每一份数据常规有两个副本，这也使得存储利用率仅为1/3，每TB数据都需要占用3TB的存储空间，相当于增加200%的存储开销，而且数据的读写也给网络带宽带来了很大的压力，而且随着数据量的增长，复本带来的开销越来越大。在保证可靠性的前提下如何提高存储利用率已成为当前HDFS应用的主要问题之一。针对这些问题，英特尔和Cloudera开始引入纠删码（Erasure Coding，EC）技术，在保证数据可靠性的同时大幅降低存储开销。相关代码已经进入trunk，并计划在2.9或3.0版本中发布。

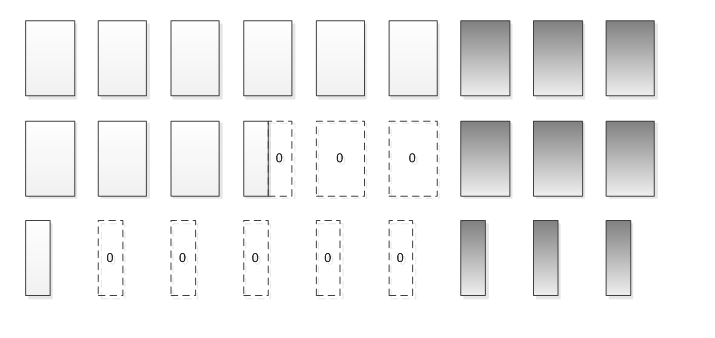
# 1.EC，Erasure Coding

Erasure coding纠删码技术简称EC，是编码容错技术，其起源于通用传输领域用于数据传输中的数据恢复，其通过在原始数据中加入新的校验数据，使得各个部分的数据产生关联性，当一部分数据丢失时，通过剩余的数据和校验数据计算出丢失的数据。

其核心思想是将K块原始数据原始通过一定的编码计算，得到M块校验元素，对于K+M块元素，当其中任意的M块元素出错（包括数据和校验数据），均可以通过对应的重构算法恢复出原来的K块数据。生成校验码的过程被称为编码（Encoding），恢复丢失数据块的过程称为解码（Decoding）。RS（Reed Solomon）是存储系统较为常用的一种纠删码。如下图所示：



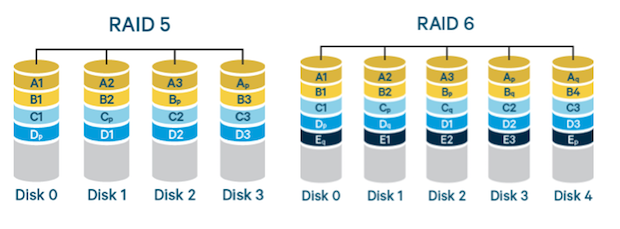
RS(K,M)最多容忍M块数据丢失，在EC模式下的文件，构成的基本单位为块组（Block），由一定的数据块加上生成的校验块一起构成。RS(6,3)为例，每个块包含1-6个数据块，以及3个校验块，下面是三种不同类型的数组以及其编码：



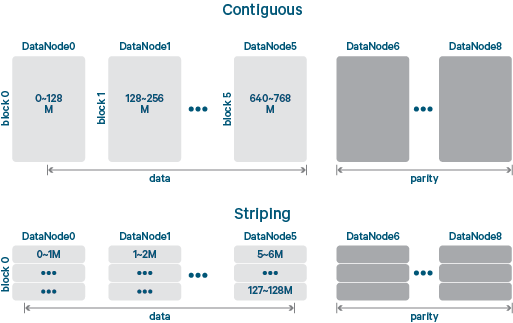
RS算法：https://blog.csdn.net/shelldon/article/details/54144730

# 2.EC in Storage System

EC纠错码技术在本地存储系统中是常用技术，典型的使用是RAID-5和RAID-6：RAID-5使用XOR纠错码，容忍单盘失败，RAID-6使用Reed-Solomon，容忍两个校验cells，如下图所示：



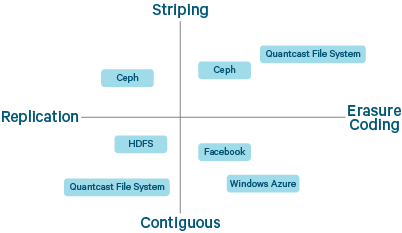
在分布式存储系统中，将大文件切分成多个固定大小的逻辑块，在集群中逻辑块映射到存储块，逻辑块和存储块之间的映射方式有两种连续布局和条形布局，如下图所示：



在连续布局模式下，数据依次写入存储块，当该块写满后再写入下一块，在一些分布式文件系统如QFS和Ceph使用连续布局的方式；条形布局模式下，每个Stripe都多个存储Block，每个Block由多个若干个相同大小单元构成的序列，数据依次写入Stripe的各个单元中，当一个单元写满后，写入下一个Block的单元。当一个Stripe的数据单元写满后，计算出校验单元并输出到校验块对应的Cell。

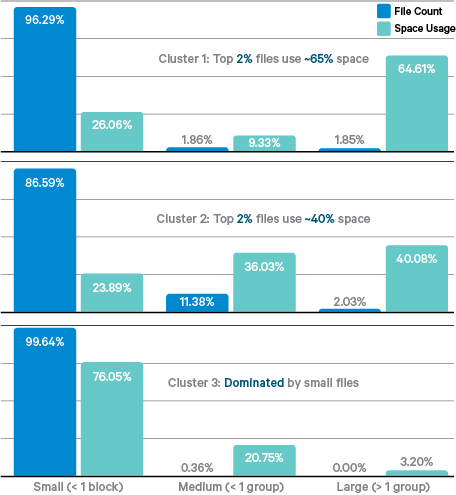
# 3.EC in HDFS

当前很多的分布式存储系统使用复本来保证数据的可用性，因此如下图，基于复本/EC和连续/条形布局的两个维度，形成存储系统的一个象限：



目前HDFS主要使用连续布局的方式将大数据分块，通过复本放到多个DN上来实现数据的可用性。另外一种方式是采用条形布局的方式将数据分发到多个DN上，如QFS,Ceph及Lustre等。

Hadoop分布式文件系统存储的核心是大文件，但是随着Hadoop中运行的应用越来越多，其存储的小文件也逐渐增多，所占的存储空间可以占到36%-97%，如下图是Cloudera对集群的调研：



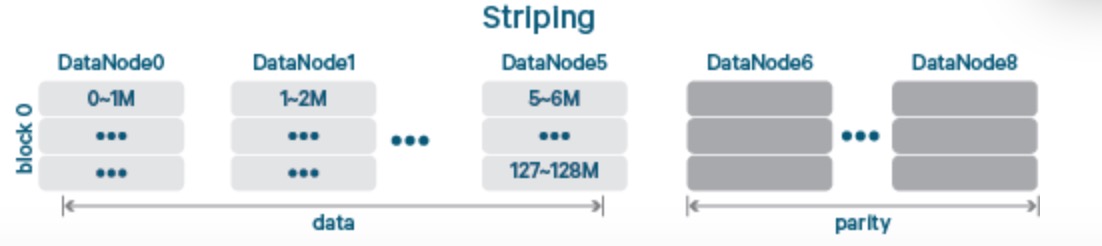
而且随着Hadoop的发展，有多种不同类型的负载，从批处理的MR作业、交互式作业、Impala延迟敏感的查询到HBase数据存储，数据类型较大，每种作业所处理和写入的文件大小不一致，产生的小文件也是越来越多。在运行过程中，大量数据都变成冷数据，3复本所带来没有必要的存储开销。在HDFS中，使用EC优劣势如下：

1. 优势，解决分布式存储系统复本策略带来的存储空间翻倍的问题，节约存储成本
2. 劣势，数据丢失后，需要从其他节点读取数据和校验块进行数据恢复，带来网络带宽和CPU资源的消耗。

所以概括的将，EC适用于冷数据的存储，在冷数据集群中大量的数据长期不会被访问，而且存储量大，采用EC技术可以减少磁盘使用量。而且冷数据集群稳定，耗资源少，一旦进行数据恢复，对集群不会造成较大的影响。在HDFS中，布局方式的对比如下：

1. 连续布局实现起来容易，但只适合较大文件，另外如果让Client端直接写一个连续布局文件需要缓存足够多的数据块，然后生成校验块并写入，如RS(6，3)为例，blockSize=128M为例，Client端需要缓存1.12G的数据，这点决定了连续布局的文件更适合由普通文件转换而来。
2. 但是HDFS中小文件很多，使用条形布局不会有上面带来的问题，由于一个Strip的Cell较小（64k或者1M）,因此无论文件大小，条形布局都可以为文件节省空间。Client端写完一个Strip的数据单元后就可以计算出校验单元并写出，因此Client端缓存的数据很好。但是条形布局的缺点是影响一些位置敏感任务的性能，因为原先在一个节点上的一个块被分散到多个不同节点

但是基于上面集群文件的分析和布局方式的优缺点，HDFS EC优先考虑支持条形布局，如下图所示：



例如6blocks的文件使用3复本，需要18个Blocks的磁盘空间，但是使用EC(6 data, 3 parity)实现，只需要消耗9 block的磁盘空间。上图使用的是RS(6,3)，前面从DN0-DN5共有6个数据块，DN6-DN8是校验块。

# 4.HDFS EC的实现

由于HDFS的内部逻辑已经很复杂，所以HDFS EC的分成三个阶段：

阶段1：HDFS EC使用条形布局（Striping Layout）的文件读写方式，如果一个文件丢失，后台能够检查并恢复；如果在读的过程中丢失，能够立即解码出此时的数据从而不影响文件的读操作。相关JIRA:HDFS-7285，初步将在hadoop 3.0中发布

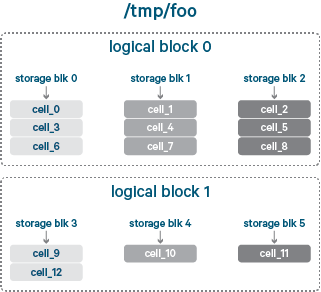
阶段2：支持将多复本文件转换成连续布局EC之间的相互转换，相关JIRA:HDFS-8030，正在进行中

阶段3：将编解码器插件化，用可以指定文件所使用的解编码器

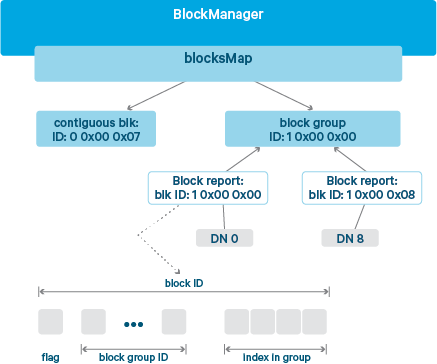
目前阶段1已基本完成，在HDFS中原生支持的复本Schema，HDFS Block丢失后可以在线或者离线恢复（比较简单，仅完成复本间的复制）。EC Schema也需要支持在线和离线的数据恢复：

1. Passive Recovery，在EC中只有一个复本，读取Block的过程中发现数据丢失，需要重构Block，数据的恢复会影响请求的延迟
2. Active recovery， HDFS NameNode周期的获取丢失或者损坏的bock，然后进行恢复

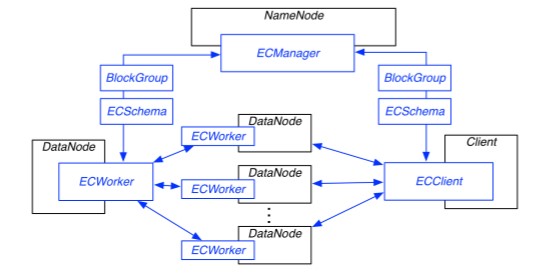
下面介绍其实现，在HDFS内部逻辑中使用的是连续块布局的方式，为了支持条形布局，需要将逻辑块和存储块的概念分开，逻辑块表示文件的一个逻辑byte数组，存储块为DataNode上的数据块。如下图：



文件/tmp/foo逻辑上分成13个striping cells，从cell\_0到cell\_12。逻辑Block0表示Cell 0~8，逻辑Block 1表示Cells 9 ~ 12。Cell 0,3,6构成物理存储块，以块的形式存放在DataNode上。为了支持逻辑块，需要在NameNode上增添一个Login Block 到Storage Block的映射Map。为了减少Block的维护负担，实现了hierarchical block naming protocol，当前HDFS在数据block创建的时候，就分配一个Block ID，但是HBNP将block ID分成2-3部分，如下图所示：



每个Block ID开始于一个Flag，标识该Block为连续或者条形Block，对于Strip Block，ID包括两个部分，中间部分为逻辑ID，末尾部分为逻辑Block包含的物理Block。Storage Block ID可以通过索引执行logic block。HDFS EC的主要工作就是维护该Block的映射关系，其系统结构图如下所示：

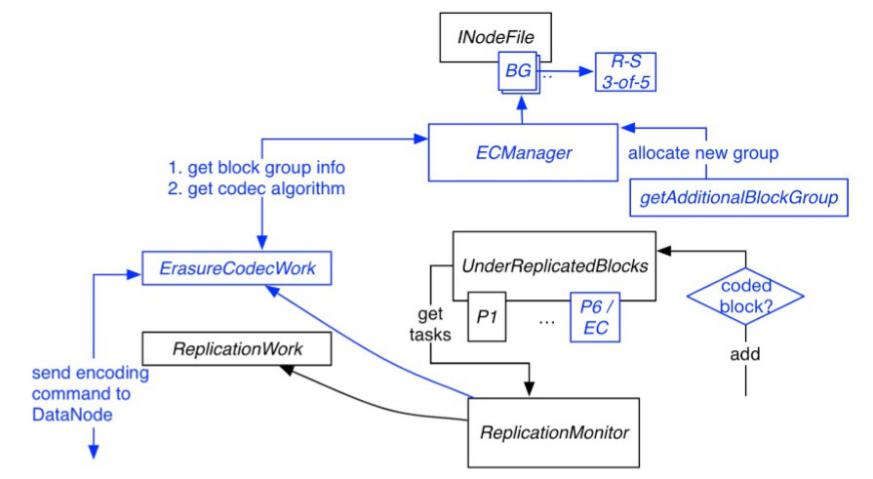


1. NameNode Extension

在EC模式下，构成文件的基本单位为块组，因此首先需要考虑的是如何在NameNode里保存每个文件的块组信息。一种比较直接的方法是给每个块组分配一个块ID，同时用一个Map来记录这个ID与块组信息的映射，每个块组信息包含了每个内部块的信息。对小文件来说，这种做法将增加其在NameNode中的内存消耗。以RS（6,3）为例，如果一个文件比6个块略小些，那么NameNode必须为它维护10个ID（1个块组ID、6个数据块ID和3个校验块ID）。在小文件数目占优的情况下，NameNode的内存使用将面临考验。

一个块ID有64位，这里将第1个位作为flag来区分块的类型：如果为1，则为EC块（条形布局的EC块，连续布局将在第二阶段考虑）；如果为0，则为普通块。对EC块来说，会将剩下的63位分成两部分：最后的4位用来标识内部块在块组中的位置，前面的59位用来区分不同的块组。块组ID等同于第0个内部块ID，其他的内部块ID可由块组ID加上其在块组中的位置索引得到，比如第0个内部块ID为0xB23400（也即块组ID），那么第3个内部块的ID为0xB23403。由于只是用最后4位来区分一个块组中的内部块，因此对一个块组来说，系统目前支持最多16个内部块。

尽可能地利用HDFS当前的机制来实现对块组的支持。块组依旧用类Block来表示，其中的三个成员：blockId代表块组ID；numBytes代表块组大小，即所有数据块大小之和，不包括校验块的大小；generationStamp代表块组的生成时间戳，所有内部块共享块组的时间戳。这里可以根据块组的Block对象得到内部块的Block对象：blockId由块组ID加上该内部块在块组中的位置索引；numBytes可由块组大小和该内部块位置索引计算出来；生成时间戳等同于块组的时间戳。每个内部块所在的DataNode信息会存储在其所属块组在BlocksMap中对应的BlockInfo对象中。当一个块被DataNode报告给NameNode时，NameNode可以通过其块ID判断该块的类型，如果为EC块则将最后4位清零便得到块组ID。通过这种方式，我们只存储了一个块组ID，大量内部块的ID通过计算得到。内部块的大小和时间戳也可由块组信息得到，因此大大减少了NameNode内存的占用。

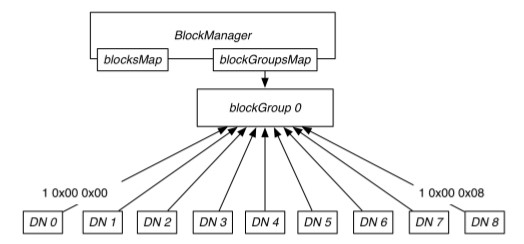


当client写满一个块组或刚开始写数据时，它向NameNode申请一个新的块组。新的块组保存于INodeFile中并返回给client。新块组包含各个内部块的DataNode信息，它们由BlockPlacementPolicy确定。这里需要各个内部块尽可能的分布在不同的DataNode或机架上，以免造成多个内部块同时丢失而加大数据丢失的风险。因此，HDFS EC需要使用符合自己要求的BlockPlacementPolicy。

NameNode中有一个守护线程ReplicationMonitor，它会周期性地执行数据块的备份和删除任务，这些任务由类UnderRepliationBlocks和InvalidateBlocks来维护。这种方式也非常适合EC任务，包括丢失块的恢复，块在多备份模式和EC模式之间的转换等。

UnderReplicatedBlocks类负责对副本数不足的块进行复制。它包含多个优先级队列，用于区分不同复制任务的紧急程度。我们可以将一个块组的副本数定义为其数据块和校验块数目之和，如果其中的一个内部块丢失，就将其副本数减1，这样其副本数就不足，就可以放入到UnderReplicatedBlocks的某一个队列中。可以根据丢失的内部块的数目来决定加入到哪个优先级队列中。当选定的DataNode传来心跳时，NameNode向该DataNode发送一个BlockECRecoveryCommand，DataNode接收到该命令将启动一个恢复任务。需要注意的是，当发现一个块组缺损后未必立即启动恢复任务，因为恢复任务会消耗大量的网络带宽，以RS（6,3）为例，承担任务的DataNode需要读取6个内部块用于解码工作。如果一个集群每天有1%-2%的节点宕掉的话，立即启动恢复任务可能会耗尽系统的带宽。因此，恢复任务的执行需要配合一定的策略，例如，优先执行缺损厉害的块组，每天限定恢复任务的数量，在系统空闲时启动恢复任务等。

BlockManager维护EC的BlockGrousMap，如下图所示：

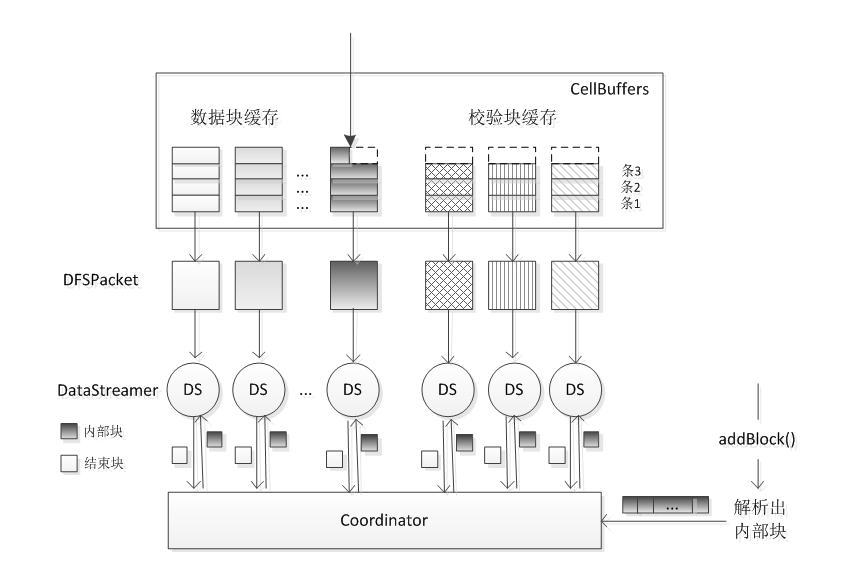


1. Client端Extensions

用户在写一个文件前可以先指定该文件为EC模式，也可以对一个文件夹指定EC模式，然后所有写入该文件夹的文件都默认为EC模式。HDFS Cleint通过输出流DFSOutputStream向文件系统中写入数据。DFSOutputStream的实现较为复杂，为了能够对其进行功能扩展，我们对该类进行重构，将内部类DataStreamer和Packet（重命名为DFSPacket）独立出来，使得各个类都有清晰独立的功能，从而为实现EC模式下的写操作提供便利。

当写一个条形布局的文件时，需要将数据分散到多个DN上，为此实现DFSOutputStream的子类DFSStripedOutputStream，它拥有多个并发DataStreamer，下图给出了

DFSStripedOutputStream的内部工作原理：



数据以条形布局的方式写入到CellBuffers中。CellBuffers拥有多个缓存，每个缓存对应条中的一个单元，也对应着一个DataStreamer。当一个条中的数据单元写完之后，DFSStripedOutputStream会立刻计算出校验单元并写入到校验块缓存中。当一个缓存中的数据能够装满一个DFSPacket时，该缓存就将数据装入一个DFSPacket并传给该缓存对应的DataStreamer。当数据写完准备关闭文件时，最后的一个条可能数据单元没有写满，这时需要对数据单元补零后生成该条的校验单元并写入校验块缓存中， 然后生成最后的DFSPacket并将它们发送至各自的DataStreamer。

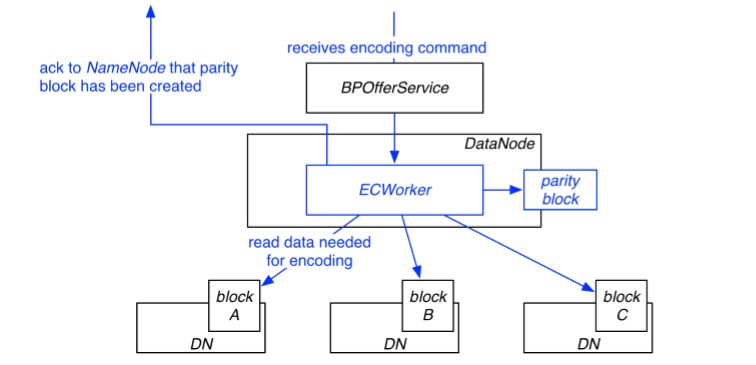
在写一个新的块组前，DFSStripedOutputStream会向NameNode申请分配一个新块组。NameNode会返回一个LocatedBlock对象，该对象包含每个内部块的DataNode信息。DFSStripedOutputStream会解析该对象，生成出各个内部块然后发送给Coordinator。Coordinator负责各个DataStreamer与DFSStripedOutputStream的协调工作：DataStreamer会等待Coordinator传递给它一个内部块用于创建数据流通道；当DataStreamer开始工作后，DFSStripedOutputStream便会在Coordinator上等待各个DataStreamer的工作结果；DataStreamer写完一个内部块后会向Coordinator发送一个结束块，以汇报此次工作的结果（如写入多少字节的数据）；Coordinator搜集到所有的结束块后汇报给等待在其上的DFSStripedOutputStream；DFSStripedOutputStream据此来决定后续步骤，如果太多的DataStreamer失败，则结束写操作并返回，否则转向处理CellBuffers中新的数据。

DFSStripedOutputStream在写过程中可以容忍一定数目的DataStreamer失败。以RS（6,3）为例，如果在写一个块组时，失败的DataStreamer不超过3个，那么失败的内部块在以后读取时被计算。 当下一个块组到来，失败的DataStreamer又可以重新开始写一个新的块

client读一个条形布局文件的逻辑要相对简单。由于数据源分布在多个DataNode上，因此在进行读的时候需要连接多台数据块所在的DataNode。 读操作的扩展功能由DFSStripedInputStream实现，它继承了DFSInputStream。DFSStripedInputStream以条为单位进行读取。client在写的时候，如果一个块组中只有较少的内部块写操作失败，client会继续写下去。因此，client在读的时候就会遇到个别数据块丢失的情况。为了能使读操作进行下去，client需要连接一定数目的校验块，读取相应的校验数据并通过解码来得到丢失的数据。这种情况下，client需要连接更多的DataNode以获取参与解码的校验数据，并且解码也会消耗client一定的CPU资源。为了减少这种情况的发生，我们需要在后台来检测有缺失的块组并进行恢复。

1. DataNode端扩展

DataNode端的扩展主要是为了实现后台对丢失数据块的解码，以及对数据块进行编码生成校验块，编码部分将在第二阶段实现。



上图展示了DataNode端的扩展，为了独立的处理EC相关的任务，我们在DataNode中添加了一个新的类ErasureCodingWorker。该类维护了一个线程池，每当有一个解码任务到来时，便会将任务交由ReconstructAndTransferBlock线程处理。ReconstructAndTransferBlock线程从若干个DataNode读取解码所需要的数据，执行解码计算，然后将恢复出来的块保存到目标节点上。一旦任务完成，ErasureCodingWorker会向NameNode发送一个确认

https://blog.csdn.net/androidlushangderen/article/details/50724917

http://blog.cloudera.com/blog/2015/09/introduction-to-hdfs-erasure-coding-in-apache-hadoop/