# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/015\_Hadoop集群中临时下线几台DataNode机器的操作步骤/笔记.doc**

一旦你这里做了一个刷新，背后会做一些什么样的事情呢？如果几台datanode机器要下线的话，你必须得把那些datanode上的block都放入待复制的队列中去，等待ReplicationMonitor线程来复制block副本到其他的datanode上去

就可以保证说，那些下线的datanode的数据都迁移到了其他的datanode上去

下线的datanode跟普通宕机的datanode是不一样的，decommissoned，下线的意思，这个机器上的datanode他可能还存在，但是的话，他是已经下线的机器

rebalancer，就会把一些datanode上的blocks给迁移到新加入集群的datanode上去，保证集群中的各个datanode负载是均衡的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/011\_对QJM集群写入等待算法的第二次JVM FullGC完成识别以及超时延期/笔记.doc**

进一步改造代码，对第二个可能发生fullgc卡顿的地点，识别出来，做一个超时时间延期处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/009\_对QJM集群写入等待算法的第一次JVM FullGC完成识别以及超时延期/笔记.doc**

写代码识别到第一次fullgc的发生，以及进行超时时间延期的处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/010\_分析QJM集群写入等待算法中的第二个JVM FullGC可能发生地点/笔记.doc**

FullGC可能发生的第二个地方

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/016\_DataNode机器下线指令：refreshNodes背后的源码流程分析/笔记.doc**

细细的分析一下，下线机器的话，refreshNodes，背后的源码流程，仔细的给大家来分析一下，这块的源码，会直接跟hdffs集群容错的源码是有关系的，副本任务生成算法，会直接用到机器下线过程中设置的一些状态

仔细的分析一下，ReplicationMonitor的副本任务生成算法

有一个致命的漏洞：一次性下线多台DataNode机器时，有可能会导致部分Block数据永久性丢失，副本任务生成算法中的缺陷和漏洞

源码上的优化，副本任务生成算法，保证说一次性下线多台datanode的时候，也是可以不让任何block数据丢失的

做一下场景的模拟，来推演一下优化后的代码的运行流程，确认我们的代码优化是生效的

一旦你要下线一批datanodes，你先不要直接手动停止那些datanode，先在一个exclude文件里配置那些机器，然后执行refreshNodes指令

你一旦下线了一批机器的话，有可能会将一些block放到复制队列里去，也有可能没有复制block，就直接将这个状态设置为decommissoned，decommisson\_inprogress，这批机器的状态就会变为上述的两个状态

下线的状态，即使他的状态变为了decommissoned以后，其实有可能这个datanode还是可以正常工作的，因为那个datanode进程此时可能还没有被你给手动停止掉

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/005\_QJM集群写入等待算法的入参梳理以及完整算法流程分析/笔记.doc**

细致的分析QuorumCall.waitFor算法

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/013\_重新回顾HDFS数据容错机制的核心原理以及运行流程/笔记.doc**

进入到优化的第二个点：一次性下线多台DataNode机器的时候，可能会导致部分Block数据永久性丢失

hdfs数据容错机制重新回顾一下，就怕有的人，回过头把源码再去看看

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/001\_HDFS HA架构以及edits log写QJM集群的机制回顾/笔记.doc**

hadoop源码的二次开发、系统优化以及问题修复

后面可能会不断的增加内容，刚刚讲完hadoop hdfs的源码，第二个课程，刚开始首先主要就是对hadoop hdfs做源码的二次开发

hadoop -> hdfs（分布式存储），yarn（分布式资源调度），mapreduce（分布式计算）

还会讲解yarn和mapreduce的源码，会放到后面有一个离线数据那一块的大项目，项目驱动，来讲解这两块的源码

我们可以回过头来补充这个hadoop源码二次开发的课程，对yarn和mapreduce进行二次开发，1个月左右的时间，对hdfs进行源码二次开发

从源码级别解决的hdfs的2个问题

（1）NameNode如果出现FullGC卡顿时间过长（卡顿二三十秒），可能会导致edits log写入QJM（JournalNodes集群）算法会出现异常的失败，进而导致NameNode宕机

（2）在手动一次性下线多台DataNode机器的时候，block副本复制的算法有缺陷和漏洞，可能会导致部分数据永久性丢失

从源码级别去分析可能存在的问题，然后改造源码，提升hdfs系统的稳定性

源码二次开发的课程有什么好处呢？如果你去看一些稍微要求高一点点的大数据岗位的招聘的话，资深大数据岗位，30k~40k，40k+的专家岗位，其实都是源码有要求的，hadoop、spark、kafka，这些流行常见的大数据系统的源码

深入阅读过开源项目源码，或者是对开源社区有过贡献的人，或者是能够对开源项目进行源码二次开发的人，优先

对开源项目做贡献的话，还是有点麻烦的，等别人来审核你的patch

你作为公司内部的核心的大数据工程师的话，你是否对你们公司内部维护的集群做过源码级别的二次开发和定制，修改源码，提升系统稳定性、性能、修复问题、优化

在国内，大数据技术好的人很少

1个多月的时间，来做第一个大数据的大项目，hadoop源码二次开发，很多个地方都要修改他的源码，提升他的稳定性、性能

创建目录，会对应元数据的变更（Filesystem Namespace，文件系统命名空间），维护他的一些文件目录树，edits log，修改日志

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/002\_edits log写入QJM集群的源码再次梳理以及核心组件回顾/笔记.doc**

再次重新过一下写一条edtis log日志的话，比如说，你的NameNode在创建一个目录的时候，肯定会写一条edits log日志，这个源码级别的执行流程，回顾一下

我们比较关心的是 edits log如何刷入QJM集群

QuorumCall是一个核心组件，他里面负责封装了一个非常关键的算法，QJM集群写入等待算法，如果有3个journalnode，你就必须写入quorum数量的journalnodes节点，才算是一条edtis log写成功了

quorum数量 = 3 / 2 + 1 = 2

必须至少将edits log写入2个journalnodes节点才可以，算作是本次edits log写入QJM集群成功，在成功之前，都会在这个写edits log到QJM集群的过程里卡住，阻塞等待，QuorumCall就封装了上述算法

writeTimeoutMs，你最多会等待多长时间写入edits log到大多数的journalnodes中去，默认的值是20s，QuorumCall.waitFor()，里面就封装了QJM集群写入等待算法，在这个算法里，他会在指定的超时时间范围内（20秒），等待一条edits log写入到大多数的journalnodes节点中去

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/007\_自己手写一个完整易用的JVM FullGC卡顿时间流逝监控组件/笔记.doc**

改造源码，比较核心的一块东西，就是最主要的一个问题就是JVM FullGC导致的时间卡顿，组件就是来监控JVM FullGC卡顿导致流逝掉了多少时间，组件

就可以用这个组件来改造那个QuorumCall.waitFor()算法

hadoop 2.7.x系列里面作为一个基础的工具类，放在了hadoop-common包下，org.apache.hadoop.util包下

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/018\_Block永久丢失问题：一次下线所有副本所在机器背后的问题/笔记.doc**

之前的话呢，已经对源码进行了比较详细的复盘

假设我们现在集群里有20台机器，部署了一个datanode，有20个datanode在运行

假设我们现在有一个block，3个副本，分别在datanode01、datanode09、datanode13，三台机器上放着，此时，天公不作美，突然间，集群里下线3台机器，而且刚好就是datanode01、datanode09、datanode13

三台机器上的datanode进程先别停止掉

在namenode的exclude文件中，可以放上三台机器，运行refreshNodes指令

要么是datanode是在decommison\_inprogress，decommissoned状态，假设说，三台datanode此时都变成了decommissoned状态，但是他们所在机器上的datanode进程还在运行着，此时还没停止

刚好这个block是放在复制队列里的，此时需要对这个block生成一个复制任务

一定会对这个block选择一个soruce datanode

就会拿到这个block对应的各个datanode，比如说刚好是datanode01、datanode09、datanode13，依次来进行判断

此时datanode09突然自己宕机了，导致block被放入了复制队列里

然后此时你还手抽筋在下线datanode01和datanode13两台机器，直接就会导致block在选择source datanode的时候，直接无法找到任何有效的datanode

就有可能会导致你手工下线机器的时候，导致部分block永久丢失，因为没有任何副本存在了，也没复制副本，导致数据出现破损

极端情况，但是这些情况是在生产环境里有可能会发生的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/004\_在指定时间内写入QJM集群失败是如何导致NameNode宕机的/笔记.doc**

从源码级别来看看，如果QJM集群写入等待算法失败的话，抛异常的话，是如何导致NameNode宕机的

QuorumCall.waitFor()算法极为的重要以及核心，如果一旦失败，认为是journalnodes集群故障了，无法写入edits log到里面去，直接就会一层一层往上抛出异常，然后到最外层，直接导致System.exit(1)异常退出

NameNode宕机

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/017\_ReplicationMonitor的副本任务生成算法再次详细解读/笔记.doc**

下线机器的时候，人家的状态要么是正在下线中，已下线，但是那些datanode节点可能还是在运行的，还是可以正常工作的，有可能会将一些block加入复制任务队列中去

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/008\_分析QJM集群写入等待算法中的第一个JVM FullGC可能发生地点/笔记.doc**

分析一下，在哪个地方会发生fullgc导致的卡顿

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/019\_优化Block副本缺失复制算法：下线机器作为兜底候选机器/笔记.doc**

其实我们想要实现的效果是什么呢？

有几台机器是decommissioned状态，但是datanode进程还在运行着

此时，如果说在选择source datanode的时候，无法选择到一台正常合适的datanode，我们就从那些decommissioned状态的机器里随机挑选一台，作为source datanode，他还在正常的运行和工作中

下线的机器，是有对应的block副本的

if(node.isDecommissioned()) {

if(decommissonedSrcNode == null ||

DFSUtil.getRandom().nextBoolean()) {

decommissonedSrcNode = node;

}

continue;

}

if(live == 0 && srcNode == null && decommissonedSrcNode != null) {

return decommissonedSrcNode;

}

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/014\_HDFS数据容错机制的核心源码流程再次快速走读以及回顾/笔记.doc**

集群容错机制的源码，给回顾一下

注意一下里面的核心组件，我们下讲开始，来给大家讲一下手动下线机器的做法，以及背后的源码的流程

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/006\_NameNode FullGC是如何导致QJM集群写入等待算法异常报错的/笔记.doc**

非常关键的一讲了，然后来分析一下如果NameNode突然发生长时间的FullGC卡顿的话，是如何导致QJM写入等待算法异常报错的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/003\_NameNode稳定性优化：FullGC导致QJM集群写失败后异常宕机/笔记.doc**

3616118202

简单给大家带出来一个主题，梳理和回顾了一下edits log机制，图原理，源码，edtis log是如何写QJM集群的

QuorumCall.waitFor()，QJM集群写入等待的算法

但是兄弟们，别忘了，如果是Java，JVM，杀伤力的问题，就是FullGC卡顿，会导致大量的线上系统的异常，NameNode如果一旦不小心出现了FullGC导致超过20秒的卡顿之后，就会导致一个问题

QJM集群写入等待算法会没执行完，直接异常退出

JVM FullGC是什么都不知道？半路出家来学的大数据，起码是java开发经验3年以上，最佳的状态是5~8年的java系统开发的经验，技术比较好的，同时学大数据的技术，精通大数据技术的速度是很快的，2年时间，就可以快速的达到大数据专家的水平

JVM？FullGC，垃圾回收，3~5年经验的java工程师必备的知识储备

hadoop、spark、kafka，java、scala，jvm

垃圾回收的问题，比如说你代码里有一些对象，用完了，没用了，垃圾，jvm就要回收这些垃圾。如果比如说hadoop namenode这种java开发出来的系统，他会内存里驻留大量的数据，导致垃圾很多

jvm fullgc完整的垃圾回收的时候，可能会耗费大量的时间来回收对象垃圾

也许会耗费20秒，甚至是几分钟的时间来回收jvm的内存垃圾。在回收的过程之中，stop the world，回收垃圾的时候整理内存空间，不让你的所有工作线程运行了，避免他一边整理内存垃圾，一边还再制造内存垃圾

jvm，垃圾回收，stop the world，导致你所有的工作线程全部 停止，专门留下空闲的时间给垃圾回收的线程来回收垃圾

一旦说jvm fullgc长时间卡顿，导致QJM集群写入算法抛异常了之后，就会导致NameNode直接宕机

hadoop 2.6.5版本，线上运行的过程中，有一定的概率会在生产环境中有这个问题发生，不是说一定会发生，是有一定的概率会发生，尤其是如果你的集群是特别大的规模的话，导致集群规模大，数据量大，namenode里的内存数据量也打

fullgc卡顿的时间长，如果达到了一个极致的临界条件的话，就有一定的概率导致fullgc卡顿引发namenode突然宕机，你都不知道是怎么回事

源码改造，提升namenode系统稳定性，fullgc如何发生，不会导致QJM集群写入等待算法异常，不会引发namenode突然宕机

hadoop 2.7.4版本里做的一个源码修复，提升了namenode的稳定性，基于我们hadoop 2.6.5这个版本的源码来做改造，hadoop 2.8，hadoop 2.9中提取一些比较关键和核心的系统优化的部分，拿出来在课程里来讲解

初步的掌握源码二次开发的能力，如何去修改源码，有了源码修改的项目经验，面试，杀伤力极大，除非是我挖我的一些朋友来我这里，都是各大公司技术非常好的专家，一般从市面上招，这样的有源码二次开发能力的大牛，几乎找不到

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/020\_复盘Block永久丢失场景以及优化算法的运行机制如何解决问题/笔记.doc**

假设block有3台datanode，此时其中1台宕机，导致block进入了复制队列，另外两台都被手工下线了，变成了decommissoned状态

解决了另外一个问题：下线datanode的时候有一定概率，导致部分block永久性丢失

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/001~020资料/012\_模拟JVM FullGC场景发生以及优化后的QJM集群写入算法运行流程/笔记.doc**

模拟一下两个fullgc都发生了，此时这个qjm集群写入等待算法的运行流程

推演一下

哪怕是也许有个别细微的小bug，但是大的流程基本上就对了，hadoop 2.7.4版本修复了这个问题，基本上实现思路跟我们的是一模一样的，99%相似，稍微在延期处理的时候有一点点不一样

每次et = et + millis（20s）

解决的问题：NameNode FullGC可能导致QJM集群写入等待算法超时异常，进而引发NameNode突然宕机

提升了NameNode的稳定性：避免说因为各种各样的异常的情况，磁盘坏了、网络故障、内存溢出、磁盘满了、JVM FullGC卡顿，导致NameNode宕机

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/028\_hdfs的block上传源码没有及时fsync到磁盘导致的数据丢失隐患/笔记.doc**

内存buffer、OS buffer、磁盘文件，fsync，flush，都给大家铺垫好了前置的知识，都结合JDK官方文档对IO流的flush的语义做的说明

默认情况下，如果OS不及时的fsync到磁盘，会在机器宕机的情况下，导致一些数据的丢失，ES举了一个例子，写入的数据不是立马fsync到磁盘，导致写入的性能很差，导致写入并发能力会很差

优先写入都是在内存buffer和OS cache/buffer，都是在内存层面来解决的，高性能的写入，高并发的写入，后续找一个时机来将buffer数据fsync到磁盘文件里去，减少频繁的磁盘读写的操作

hadoop，接收到block文件，也是上述的做法

导致一个问题，结合源码来看看，block接收完毕之后，仅仅是对IO流执行flush，导致数据可能停留在OS buffer层面

rename操作，File的renameTo操作，JDK官方文档对他的说明就是：具体如何执行依赖于底层的平台（OS操作系统），有可能rename不会成功，有可能不是原子的，有可能会因为目标文件的存在而失败

rename操作是否可以立即在磁盘层面生效，也是持有疑问态度的

flush、rename，都不一定立马会在磁盘层面生效，那么此时如果datanode刚刚接收完毕一个block，刚刚执行完flush和rename，还没来得及fsync到磁盘上去，此时就直接所在的机器就宕机了

是不是就是有一定概率会在机器宕机的时候，导致刚刚接收完毕的block就彻底丢失了

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/022\_再次重读BlockReceiver接收packet的源码回顾核心流程/笔记.doc**

回顾了大文件上传到hdfs集群分布式存储的原理

把这个里面的关键的核心源码，重新再来读一下，BlockReceiver接收到一个一个的packet然后写入到本地磁盘文件里去的这个过程

启动一个PacketResponder

无限while循环，不停的接收一个block的各个packet

（1）将packet放入一个PacketResponder的ackQueue里

（2）将packet推送给下一个datanode

（3）将packet写入本地blk磁盘文件里去

一个block的所有packet都接收完毕之后，就会关闭PacketResponder

finalized block文件，分析过一个block上传的过程中，刚开始在rbw目录里，代表这个block的数据正在上传中，finalize，将一个上传完毕的block的文件从rbw目录里挪动到finazlied目录里去

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/029\_数据可靠性提升：block上传完毕后立马及时fsync到磁盘避免数据丢失/笔记.doc**

修改block接收的源码，在block只要一旦接收完毕之后，就立马fsync到磁盘，避免说fsync的等待时间过长，导致中间万一机器宕机了，刚接收到block就丢失了

boolean isDir = fileToSync.isDirectory();

FileChannel channel = FileChannel.open(fileToSync.toPath(),

isDir ? StandardOpenOption.READ : StandardOpenOption.WRITE)

try {

channel.force(true);

} catch(Exception e) {

} finally {

channel.close();

}

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/024\_Java IO流中的flush操作到底能不能立即将数据刷入磁盘？/笔记.doc**

如果一个block写完了以后，主要是两个磁盘的操作：

（1）对block磁盘文件调用了一次flush

（2）将rbw目录下的block磁盘文件挪动到了finalized目录下去 -> 都是操作系统层面的事情，可能都是停留在OS层面的，还没持久化到磁盘上去

如果你别的什么都不干的话，就仅仅只是这样做一些操作，你觉得对磁盘上的文件和目录的操作，能够立即的生效吗？

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/027\_Elasticsearch中的写缓冲和不及时flush可能导致数据丢失的潜在问题/笔记.doc**

引申一下，Elasticsearch，如果不了解的话，我之前讲过一套es的课程，可以找jeverson同学去要一下，自己去看看。分布式搜索引擎，你写入数据来建立索引，别人可以通过索引来检索数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/032\_为了源码二次开发这次要透彻的100%解析内存双缓冲机制/笔记.doc**

提点一下，为了要彻底分析清楚hadoop edits log机制的一些缺陷所在，要非常详细的分析源码的细节，之前的话呢对edits log机制的源码解析稍微粗了一些，而且没有专门的为这块源码去画图

就是在一个大的图里画了一个示意图

hadoop源码二次开发的课，很多的部分是在细致的分析源码，分析源码可能产生的问题，铺垫你需要的一些基础的知识，到最后实际修改源码的代码量其实不大，而且可能代码量还比较小一些

你如果去看hadoop每个版本的源码的变更，你会发现其实每个commit的代码量的变化都不是很大，做一些细微的改动，不断的优化这块代码，就是跟你平时自己做一个项目开发，是一样的

如果你要修改一块源码的话，那你必须把前后左右的逻辑分析清楚，想清楚了，这个源码才能动手去写，不能随便上来就开始改造源码

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/026\_写入磁盘的数据停留在内存缓冲区中会因为机器宕机而永久丢失？/笔记.doc**

（1）block写完之后，会flush + rename，但是都停留在OS层面

（2）OS是不会立即落地到磁盘层面

（3）OS会找个合适的时机（不确定的），落地到磁盘层面

如果OS没有将磁盘的操作以及数据落地到磁盘层面，如果停留在OS cache/buffer的层面里，如果此时机器突然宕机、重启类似的情况，会直接导致OS层面的数据丢失，此时就会导致OS内存里的数据就全部丢失

你之前flush出去的那些磁盘文件的数据，就没了，对磁盘文件做的rename的操作还没来得及落地到磁盘层面去，此时就机器宕机了，就会导致失败，磁盘文件还停留在rbw目录下面，没有到finalzied目录下去

刚上传完毕的block数据就可能会永久性的丢失

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/035\_一个线程开始刷新数据到磁盘之前其他线程全部强制等待/笔记.doc**

一旦bufCurrent缓冲区写满了512kb之后，一定会有一个线程修改isAutoSyncScheduled变量为true，他会释放锁然后尝试执行logSync操作

isAutoSyncScheduled = true，他的影响和效果是什么呢？

只要一个线程修改了isAutoSyncScheduled = true之后，然后他会开始尝试flush数据到磁盘上去，此时后面的线程都会卡在while循环里，不停的等待isAutoSyncScheduled标志位变为false

此时可能会有多个线程卡住，无法去写bufCurrent内存缓冲区，其实这个语义是正确的，因为bufCurrent内存缓冲区已经写满了，此时就不应该让数据写入到bufCurrent内存缓冲区里去，所以人家基于一个标志位做了一个控制

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/036\_通过巧妙的线程并发控制让最早的一个线程才有资格刷新磁盘/笔记.doc**

解释一下他这么做的原因

避免了大量的线程在bufCurrent满了512kb之后还在写入数据，避免大量的线程尝试竞争去执行logSync操作，避免一个线程在flush的过程中，其他大量的线程陷入while循环不断的等待

可能就会导致一个线程在flush数据到磁盘的过程中，可能长达几十毫秒，几百毫秒，几秒，这个过程中，有很多的线程hang死在while循环里，不能返回，不能干其他的工作，这个hang死是没意义的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/040\_双缓冲机制的致命缺陷：两块缓冲区都写满之后导致NN停止运作/笔记.doc**

这块机制的致命缺陷，就是之前已经分析过了，在这个双缓冲的机制之下，其实比较坑爹的一点就是说，如果两块缓冲区都写满了，会导致NameNode（NN），DataNode（DN），会停止工作

NameNode最核心的工作就是对整个hadoop集群的元数据进行维护，包括元数据更新以及查询，必须涉及到edits log写，如果edits log写内存缓冲都卡死了，会导致大量的工作线程直接hang死在无限的while循环里

直接会导致NameNode进程的线程短时间内全部卡死，对外的表现就是NameNode无法响应外部对他的任何请求

两块缓冲区，一块缓冲区的大小是512kb，并不大啊，万一出现说高吞吐量的时候，导致大量的edits log的写入，此时就可能会导致一个缓冲区快速的被写满，在flush的过程中，可能flush耗费的时间并不多

但是因此瞬时超高吞吐的场景下，导致另外一块缓冲区的512kb的空间被快速的写满

就会在超高吞吐量下，比如说每秒请求达到上万个元数据修改的请求的时候，可能会导致短时间内NameNode的工作线程全部hang死，无法响应外部对他的请求

万一说遇到一个大规模集群部署的分布式计算引擎，他在瞬时间超高并发来大量的创建临时hdfs文件，就可能会产生每秒上万并发的请求来写edits log的日志，可能会短时间内导致两块缓冲区全部写满

我们之前线上的集群，就遇到过这个故障，默认的缓冲区的大小确实是比较小，512kb，所以确实是遇到过这个问题，就是瞬时超高并发导致NameNode在集群使用的高峰期短暂的假死，卡死，无法响应请求

分析NameNode JVM堆栈的信息，大量的线程wait在while循环的地方

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/023\_在packet数据写入磁盘文件的错综复杂代码中找到IO flush操作/笔记.doc**

细化一下来研究一下上一讲说的那段BlockReceiver接收block的源码

BlockReceiver不断的接收一个一个的packet，不断的往本地的block磁盘文件里去写，刚开始这个block磁盘文件是处于rbw的状态，不是finalized状态

如果大家对Java IO流操作稍微但凡是有一点点了解的话，用IO流的一般通常会有一个flush的操作，很多IO流自带了一个内存缓冲区，刚开始你写入磁盘文件的数据，可能有很多是积压在IO流的内存缓冲里的

一旦你写完了一个磁盘文件之后，都有一个flush的操作，将IO流自带的内存缓冲的数据写入到磁盘文件里去

（1）BlockReceiver是如何判断出来一个block接收完毕的

（2）当他判断出来一个block接收完毕之后，他会干什么事情呢？（IO流的flush）

格局lastInBlock，布尔值，如果是空packet就说明一个block传输完毕了。如果说一个block传输完毕了之后，就会理解对本地磁盘文件的OutputStream的IO流执行flush操作，尝试将IO流缓冲中的数据全部刷到磁盘上去

OutputStream IO流的flush：IO流可能会自带一些内存里的buffer缓冲区，也就是说，你往IO流写的数据，不是立马到磁盘文件里去的，可能会停留一些数据在buffer缓冲区，是为了提升性能，减少写磁盘的频率。

如果调用了flush操作，就尝试将IO流缓冲区的数据flush到磁盘里去。flush的语义，如果IO流将一些数据缓冲在内存缓冲区里的话，此时一旦调用flush操作，就必须立马尝试将这些数据刷到目标地点去，比如磁盘文件，或者网络传输。

如果你的IO流的目标地点是操作系统提供的一些抽象层，比如说操作系统里的文件，操作系统提供的一套抽象文件系统，不是说具体特别底层的硬件上的磁盘文件，此时如果你执行flush操作的话

他仅仅是将这个IO流缓冲区里的数据交给操作系统来处理，但是操作系统什么时候将这个数据真实的写入到磁盘文件里去，这个是不确定的。

最后就是finazlied操作，将rbw目录下的文件挪动到finazlied目录下的某个子目录里去

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/030\_没有100%的可靠：机器宕机时是否依然存在数据丢失的问题/笔记.doc**

在接收完毕一个block之后，仅仅flush + rename，有可能机器突然宕机，导致os buffer里的数据丢失了，此时还没fsync到磁盘文件上。hadoop 2.7.4里面，做了一个修复，我带着动手来写了代码

如果识别到一个block传输完毕了之后，在finalizeBock的操作中，就可以将这个block磁盘文件从rbw目录下挪动到finalized目录下去，挪动完了以后，直接基于java NIO的API，FileChannel，对rbw目录和finazlied目录都做一个fsync操作

将这两个目录中的数据，都及时的进行fsync，避免说刚刚接收完毕了一个block，此时机器突然宕机呢？没及时fsync，就会导致block数据丢失。此时我们尽最快的速度执行了fsync的操作，就可以尽可能避免机器宕机导致的block数据丢失

也就是哪怕你现在已经尽快的在fsync操作了，但是只要你还没fsync的时候，机器宕机了，还是可能会导致block数据的丢失。fsync，尽可能快的对磁盘目录进行fsync，缩短执行fsync操作的等待周期

尽可能的将数据丢失的可能性降低到了最低

hadoop 2.7.4这个修复，已经很近了，大概这个修复在2017的三四月份，四五月份，才修复掉的，尽可能降低丢数据的概率，100%不丢，不断的完善hadoop这个系统的可靠性，健壮性，稳定性

挑选了比较核心的一些源码改进拿出来给大家来讲，不太关注hadoop每个版本做的源码的修改，其实你到github上去，找人家的pull request的历史，里面有各个版本的代码往主干代码分支合并的记录

你可以看到每个版本的代码，都做了哪些修改，commit，每个commit就是在修复一些对应的bug，优化稳定性，提升可靠性

很多的版本，他大部分修改的代码，都是一些很琐碎的一些，优化一些字符串的处理，加一些日志的打印，或者做一些特别简单的修改，我挑选了比较核心一些commit，对系统的稳定性、可靠性有较大提升的commit

拿出来跟大家来分析和做源码的二次开发

就是edits log那一块，暂时不是hadoop官方做的改进，但是是我们自己对hadoop二次源码开发的时候，做自己的版本的时候，做的一个改进，跟大家分享一下

hadoop 2.8.x版本中挑选一些比较核心的commit，hadoop 2.9.x版本中挑选一些核心的commit，来带着大家来修改源码，hadoop 3.x那个生产用的较少，还不是特别的稳定，暂时我先不涉及到他

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/025\_操作系统层面的磁盘缓冲区以及无法预估的fsync操作/笔记.doc**

无论是对一个文件的IO流执行flush，还是磁盘文件的重命名（挪动目录），可能都是停留在OS层面的一个东西，还没落地到磁盘层面去

终归会有一个机会，OS会执行fsync操作，就会将OS层面的磁盘缓冲区里的一些数据全部刷入到磁盘层面去

一旦OS找了一个合适的时机执行了fsync之后，就会把你对block磁盘文件写入的数据，以及你对磁盘文件做的rename操作，都会直接强行落地到磁盘层面上去

其实非常简单，就是为了提升性能，刚开始的时候，你对磁盘做的一些操作，都由OS来托管，可以选择在内存缓冲里做一些操作，不是立马落地到磁盘的操作。这样的话就避免频繁的执行磁盘读写

磁盘的读写是很慢的，所以OS通过这样的方式，避免频繁的磁盘读写，优先在一些内存缓冲，OS cache，内存数据结构，来执行一些操作，后续找一个合适的时机再把一些操作落地到磁盘读写

减少磁盘读写的操作，提升你的整体的性能

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/041\_瞬时高并发请求优化：内存双缓冲大小的可配置化以及按需扩容/笔记.doc**

如何解决这个问题呢？非常简单，根据你的线上hadoop集群的负载，将内存双缓冲区的大小适当的加大，空间换时间，一块缓冲区原来是512kb，0.5mb，10倍，5120kb，5mb，把一块缓冲区的大小扩大100倍

一块缓冲区的大小是50mb，两块缓冲区加起来也不过区区的100mb而已，NameNode都是在高配置物理机，内存给的尤其大，32核128G，内存缓冲区多给一些大小又如何呢？当然这个东西

要根据你线上的情况，慢慢的调节和尝试，512kb偏小

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/037\_一个线程开始刷新磁盘后其他线程开始写数据到另一块缓冲区中/笔记.doc**

只有一个线程可以修改isAutoSyncScheduled标志位，然后开始尝试logSync刷数据到磁盘里去，其他的线程都会因为isAutoSyncScheduled标志位的阻挡，在一个while循环里卡住，无法写内存缓冲

分段加锁的好处，就在于说什么呢，就是flush到磁盘的过程是不持有锁的，这样的话呢就可以让后面的线程不断的将数据写入到内存缓冲的另外一块缓冲区里去，然后冲在最前面的线程就会将bufReady缓冲区里写满512kb的数据

不断地刷新到磁盘文件里去

但是在flush的过程中，比较耗时；其他的线程可以同时的继续将数据写入到bufCurrent缓冲区里去；两个过程完全并发的执行

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/034\_原来内存缓冲区写满512kb之后才会强制批量刷新到磁盘文件/笔记.doc**

这一讲开始来研究一下，如果随着数据不断的写入，让bufCurrent缓冲区里的数据达到了512kb的话，那么此时是会发生什么事情呢？

首先得到一个非常关键的结论：一旦bufCurrent缓冲区达到了512kb之后，有会有一个线程修改一个关键的标志位，接着尝试执行logSync操作，将写满512kb的缓冲区里的数据给刷新到磁盘文件上去

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/033\_多线程通过加锁机制依次排队快速写入数据到内存缓冲区/笔记.doc**

我们从头开始来研究一下edits log的机制

重头开始细细的梳理和分析这块机制，之前讲解的稍微粗一些，很多变量和条件的判断，都没给大家说

大量的线程依次的进入拿到synchronized锁，依次递增txid，作为自己的EditsLog的txid，全局唯一递增的事务id，然后将数据写入内存双缓冲里的bufCurrent缓冲区里，接着判断一下，bufCurrent缓冲区的数据是否超过了512kb

如果没有超过512kb，则直接返回

结论：在bufCurrent缓冲区里的数据超过512kb之前，大量的线程会非常快速的进入加锁，写数据到内存缓冲，然后立马释放，这个过程大概只会耗费微秒级别的耗时，非常快，毫秒的速度都不需要耗费到

如果说bufCurrent缓冲区写满了以后，此时就会有一个线程开始往下执行，尝试进行flush内存数据到磁盘的操作

3616118202

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/039\_一个线程刷新完磁盘过后如何唤醒下一个线程刷新另一个缓冲区/笔记.doc**

如果一个线程刷新完了一个缓冲区的数据之后，如何唤醒下一个线程来继续刷新

一个线程刷新完了缓冲区会清空掉这个缓冲区的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/031\_回头看看edits log写磁盘和网络时的内存双缓冲机制/笔记.doc**

我首先要带着大家从头到尾，从里到外，仔仔细细的，不放过任何一行代码的来重头研究一下edtis log里的内存双缓冲的机制，因为这套机制里不错，但是存在漏洞和缺陷，所以在我们自己的生产环境里，对这个东西是遇到了生产故障的

因为有那个生产故障，所以我们就后面优化了一下这块的源码，解决了这块生产的故障

内存双缓冲 + 分段加锁，可以大幅度的提升多线程并发写edits log的效率，分析这块的源码，另外在java并发课里，专门为了演示wait和notify两个东西的用法，还纯手写来实现了一下edits log机制

内存缓冲实现了edits log批量刷磁盘的机制，避免频繁的磁盘的读写，大幅度的提升了写edits log的性能和并发能力

edits log数据停留在内存buffer里还没刷入磁盘之前，此时有可能会出现NameNode万一宕机了呢？此时可能会导致那个数据会丢失。所以说人家后来引入了一个JournalNodes集群，他是先将数据写入到journalnodes集群里去

此时异步写多台journalnode节点，但是会同步的等待大多数节点都写成功，这块东西的源码，在之前提升NameNode稳定性仔细的分析过里面的源码

总有可能会丢失一点数据的，如果你要数据100%零丢失的话，那不好意思，不太适合用hadoop这套大数据的技术，可以允许一点点的数据误差和不准确，但是主要是适用于海量数据的存储、计算、分析趋势

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/021\_重新回顾一下大文件上传到hdfs集群分布式存储的原理/笔记.doc**

提升数据的容错性

按照hadoop现在的源码实现，有可能，如果一个大文件刚刚上传到某些datanode所在的机器上去，结果某一台datanode所在的机器整个宕机了，不是datanode进程挂了，是说所在的机器宕机了

机房突然停电了，机器就宕机了

可能会导致刚刚上传完毕的block的数据会永久丢失

这个问题背后的一些原因和原理，修改hadoop源码不是那么简单，咔嚓咔嚓的修改源码，理解清楚背后的一些原理和问题发生的原因，修改源码可能是比较快速的

简单回顾一下hdfs大文件上传 -> 分布式存储 -> 多副本容错 -> 上传流程容错

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/021~41资料/038\_一块缓冲区刷新磁盘的同时另一块缓冲区也写满了会发生什么事情/笔记.doc**

如果此时一个现场在刷bufReady缓冲区里的数据到磁盘文件的同时，bufCurrent缓冲区又被写满了512kb该怎么办呢？是有可能的，我们之前遇到的一个生产的故障

第二个打算刷磁盘的线程会卡在logSync里的while循环中，不能动弹，排在他之后的线程会卡在logEdit方法的while循环里，不懂动弹，不能再继续写bufCurrent缓冲区了，isAutoSyncScheduled变量是true

此时最大的一个问题就在于说，无法继续写入edits log了，导致NameNode大量的工作线程在执行元数据修改的时候，大量hang死在写入edits log这块了，无法写入，被卡在了内存双缓冲的写入这里了

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/139\_源码二次开发：在FSDatasetImpl的磁盘操作中实现性能监控/笔记.doc**

在磁盘写的操作时会监控耗时

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/133\_mapreduce分布式任务的数据拷贝源码逻辑剖析/笔记.doc**

初步看下mapreduce，CopyMapper，核心业务逻辑，数据拷贝

无非就是每个mapper任务就会针对一个hdfs文件进行拷贝，从一个hdfs集群读出来写入到另外一个hdfs集群去，大量的mapper任务一起并行执行，在多台机器上执行，把大量的文件同时拷贝和传输

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/124\_hdfs集群内部对balancer任务的自动数据限流机制/笔记.doc**

3616118202

hdfs数据限流这个topic，数据迁移

每一台机器的网络带宽是有限的，比如说现在有一些spark、mapreduce的作业在从hdfs上读写数据，那么这个数据的流动是不是会占用机器的网络带宽？如果说在执行作业的同时，你还执行balancer这样的集群内部的一些数据平衡的作业

每台机器应对spark、mr的数据读写的时候，已经有网络带宽的开销了；同时你还执行机器与机器之间的数据平衡和拷贝的任务，此时是不是也会占用网络带宽；如果说你执行的balancer的任务，一下子要平衡的数据量过大

导致balancer任务在各个机器之间的数据拷贝和传输打满了机器的网络带宽，此时会发生什么样的事情呢？

spark、mr等作业读写hdfs的时候，就没有网络带宽可以用了，此时就会影响到你的spark sql作业的运行效率，可能会导致他的运行速度变得很慢

balancer集群后台运行的任务，必须要进行限流，限制这种非核心任务对网络带宽的占用，比如说限制说balancer传输数据，每秒最多只能占用10mb的带宽，每秒只能传输10mb的数据量

每秒最多可以传输100mb的数据量，就可以保证剩余的90mb一定可以提供spark来读写数据使用，就不至于说balancer任务一下子每秒传输100mb的数据，网络带宽打满，此时就导致spark读写数据的时候没有带宽可以用了

balancer在执行的时候，在datanode之间进行数据的拷贝，datanode那边，他会自动对这种非核心的数据拷贝的任务进行限流

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/131\_使用distcp迁移数据任务开始的入口处源码剖析/笔记.doc**

distcp的源码来分析一下

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/138\_源码二次开发：将自己开发好的磁盘监控组件注册到hdfs/笔记.doc**

在FSVolumnImpl，就是每一块磁盘初始化的过程中，直接FSVolumnMetrics.create(this)，内部就可以使用FSVolumnImpl创建一个FSVolumnMetrics实例，然后放入到一个static静态的map中保存了

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/135\_找一找bandwidth参数是如何在数据迁移时限流的/笔记.doc**

全量迁移、增量迁移的源码都看过了，mapper任务做的事情就是针对机房A的hdfs某个文件读取，写入到机房B的hdfs的文件里去，限流主要是针对机房A的hdfs文件的读取来实现的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/136\_hdfs最棘手的性能问题：极为耗时的慢磁盘读写/笔记.doc**

hdfs这种集群的话，他实际上来说是最核心的一点就是磁盘，把数据放到磁盘上去，然后从磁盘上读数据，最最关键的一点就是往磁盘写数据的时候，这个性能实际上来说是尤为的关键

在hdfs集群的生产环境中，非常核心的一点，如果出现某些磁盘写性能特别的差的话，此时你就需要自己做一个告警，还得先做一个监控，监控磁盘写性能，如果磁盘写时间耗时过长的话，说明这块磁盘可能是有问题的

你可能就需要找运维工程师来进行硬件级别的运维操作，比如说把性能极差的磁盘换了，换成一块新的磁盘，hdfs集群的生产环境中，慢磁盘的问题可能是一个比较棘手的问题，有可能会出现

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/126\_datanode中的balance源码流程中的限流机制的使用/笔记.doc**

是否还记得，balance的时候，他会发送一条指令给datanode，copy block，datanode自己还会给他要拷贝数据的datanode发送一条指令

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/128\_数据迁移的几种场景：冷热集群、机房迁移、双机房备份/笔记.doc**

balance的时候，你需要注意一下他的限流这块的东西，你可以自己参数化配置一下balance的时候每台机器的带宽的占用量，均衡一下，保证数据传输不要太慢，但是也不要占用过多的网络带宽

默认是1mb，10mb

日常大数据集群运维和管理的过程中，可能会遇到的一些问题

冷热集群：把热数据放在一个集群里，冷数据放在另外一个集群里，对热数据的访问和计算走一个集群，这个集群的底层异构存储的架构配置使用的机器全都是支持内存存储引擎，冷数据集群用的都是磁盘存储

机房迁移：公司整体机房搬迁，几年才会碰到一次这种情况，大数据集群也需要整体搬迁过去，肯定是在新的机房里搭建一套集群，把老机房集群里的数据慢慢的迁移过去，迁移完毕了以后就好了

双机房备份：搞两个机房，一个机房数据需要不停的往另外一个机房的集群里同步，万一一个机房故障了，此时可以使用另外一个机房的数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/125\_BlockBalanceThrottler：hadoop限流算法的实现原理/笔记.doc**

hdfs自己内部实现的一个限流的组件，BlockBalanceThrottler

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/134\_distcp增量数据同步源码分析：文件追加写和skip跳过/笔记.doc**

增量数据同步的时候，他会skip跳过判断，append追加写的判断

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/132\_先初步简单看一下distcp如何利用mapreduce任务执行/笔记.doc**

mapreduce job是如何创建的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/129\_设计hdfs跨集群数据迁移方案的几个考虑因素和要点/笔记.doc**

机房搬迁，你的hdfs集群数据要从一个机房迁移到另外一个机房

（1）带宽：带宽过大则影响正常任务的执行，带宽过小则迁移过慢，需要限流恒定速度

你肯定是得不断的从机房A的集群里的datanode上读取数据，然后发送到机房B的datanode上去，在这个过程中，你必须得进行限流，尤其是对机房A的集群读数据以及发送数据的限流

机房A此时可能还在被spark、hive作业在运行，你数据迁移的后台任务不能占用网络带宽过大，线路

（2）分布式迁移：分布式方式迁移数据，效率更高

假设机房A的集群部署在100台机器上，你最好是直接运行分布式任务在100台机器上，每个机器一个迁移任务，每个任务就从机器上读取数据通过网络发送出去，这个过程肯定会占用网络带宽，所以限流

（3）增量迁移：周期性的解析edits log判断最近增量变化的数据，文件append和删除，都进行增量迁移

刚开始一下子把大部分的数据都迁移到了机房B的集群上去，通过分布式+限流的方式来尽心迁移，一下子肯定没法精准的把所有数据都迁移完毕的，机房A的集群还在不断的有数据变更

要多次频繁的进行增量迁移，文件append追加写、文件新建、文件删除，都是要进行多次频繁的增量迁移，每次就迁移最近刚刚变更的一些数据

直到最后，两个集群的数据可能才是差不多了

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/140\_自己实现的磁盘性能监控组件如何通过ganglia看到图表/笔记.doc**

我们这次做磁盘写性能监控，是基于hadoop metrics API来做的，所以是可以无缝的和一些hadoop监控系统进行整合，hdfs集群的监控，cdh，建议你自己尝试一下如何基于cdh来完成这个我们自定义指标的图标的展示

MutableRate就会维护一段时间内的我们添加进去的数据值，这个数据值是可以展示在图表里的，一个折线图

cdh，一般来说大数据集群的监控，可以用ganglia，你自己也可以去探索一下，如何让gangalia显示出来你自定义metric的折线图，不是技术，纯操作性的一些东西，比较麻烦一些，作业，探索

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/137\_源码二次开发：自己动手实现一个磁盘性能监控组件/笔记.doc**

磁盘写性能比较关键的一块监控，可以加在FSDatasetImpl，createRbw，createTmp

@Metrics(about = “FsVolume metrics”, context = “dfs”)

public class FsVolumeMetrics {

private static final Map<String , FsVolumeMetrics> REGISTRY = Maps.newHashMap() ;

MetricsRegistry registry = null;

@Metric

MutableRate createTmpFileOp;

@Metric

MutableRate createRbwFileOp;

@Metric

MutableRate createTmpFileTimeout;

@Metric

MutableRate createRbwFileTimeout;

private FsVolumeMetrics(FsVolumeimpl volume) {

this.createRbwFileCounter = 0;

this.createTmpFileCounter = 0;

this . createRbwFileTimeoutCounter = 0;

this.createTmpFileTimeoutCounter = 0;

String name = “fsVolume :” + volume.getBasePath();

registry = new MetricsRegistry (name);

}

public void addCreateTmpFileOp(long t ime) {

createTmpFileOp.add(time) ;

}

public void addCreateTmpFileTimeout (long time) { }

public void addCreateRbwFileOp(long t ime) { }

public void addCreateRbwFileTimeout (long time) { }

}

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/130\_基于distcp实现的双机房备份场景的数据迁移/笔记.doc**

（1）带宽限流：-bandwidth

（2）增量同步：update、append、diff

-update：同步新创建的文件

-append：同步文件追加写的内容

-delete：同步删除或者重命名的文件

（3）高性能：mapreduce分布式作业执行

hadoop distcp hdfs://nnl: 8020/foo/a hdfs: //nn2:8020/bar/foo

如果需要做两个集群之间的定时数据同步，实现双机房数据备份的话，我们就可以采用如下的方式来进行：

（1）第一次使用distcp全量同步一次数据，这个过程必须加-bandwidth限流带宽使用，否则的话就会出问题

（2）接着多次使用distcp加上-append、-update、-delete几个参数，就仅仅同步append追加写的文件，新创建的文件，被删除的文件，就可以了，然后观察两个集群的数据量是否几乎一致

（3）只要两个集群的数据量几乎一致了，就可以设置一个crontab的脚本，比如每个小时定时增量同步一次，每次都是用-append、-update、-delete几个参数增量同步最新变化的数据就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/127\_源码优化：hdfs限流组件的核心参数配置化改造/笔记.doc**

这块的源码有一个问题，时间窗口是500毫秒，硬编码，优化，改造成参数化配置

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/124~141资料/141\_对hdfs这块最后的寄语：学完课程后希望你达到的能力/笔记.doc**

hdfs两个课程都结束了

（1）cdh来搭建和部署集群，集群可视化的监控，日常可视化的运维管理，cdh文档，结合我们学习过的hdfs源码级别的原理，你自己都可以搞定了，你必须先精通这个技术，然后才能完全摸透那个cdh部署、监控、运维、管理的功能背后大致是在干什么事情

（2）资源监控，集群扩容，balance，限流，背后的源码级别的原理

（3）日常的集群异常监控，性能过差、负载过高、源码报错，你都可以从源码级别去分析问题、定位问题和解决问题，起到是一个大量练习和抛砖引玉的效果，学到的是“渔”，以后遇到类似问题，完全可以从源码级别去解决

（4）高阶功能的使用，异构存储、块缓存，类似的一些功能，如果你要使用的话，背后的原理你都已经精通了，如何使用、使用的过程有问题如何解决，我觉得你自己开始就可以尝试解决了

（5）日常的运维管理，集群数据迁移、各个团队数据权限的设置、集群安全授权和认证，基于cdh都可以做，操作性的一些东西，你完全可以自己去研究一下了，作业

对hdfs源码的分析、二次开发、问题解决、性能优化、稳定性提升、运维管理、高阶功能，讲的都差不多了，如果说hdfs彻底精通达到跟开源作者一样的水平是10分的话，在hdfs这块可以达到6分

后面的4分涉及的一些东西，你完全是有“渔”的能力，自己钓鱼，自己去解决，给你的不是“鱼”，而是给你“渔”的能力

hdfs相关的工作，遇到了一些技术难题，可以跟我讨论一下

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/067\_再次回到源码里分析为什么中止DataNode后残留大量复制任务/笔记.doc**

生产环境的问题我们已经发现了，回到源码里看看

下线的时候，核心的一个操作就是把大量的复制任务放到了neededReplications队列里去；中止下线的时候，如果你要取消大量的复制任务，就需要从neededReplications队列里挪出掉那些复制任务

中止下线，仅仅是处理那些副本数量超标的block，去删除副本。并没有处理neededReplications复制队列，并没有挪出掉那些那些对应的复制任务，所以会导致，你哪怕中止下线了

但是此时还是会在复制队列里残留大量的复制任务

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/064\_刚刚对线上集群部署下线一个DataNode结果突然要中止下线/笔记.doc**

实际你在对hdfs集群进行运维操作的时候，可能会遇到的这么一个问题。你可能刚刚对线上的集群，部署了一个操作，下线一批DataNode，刚刚执行完了refreshNodes指令之后，此时会生成大量的block副本的复制任务

你突然发现有一些DataNode是不需要下线的，或者是收到公司的通知，你执行了一个中止下线的过程

exclude文件列表加入了datanode，此时中止下线，立马把exlude里面的文件列表，中止下线的datanode从文件里删除掉，立马再次执行一次refreshNodes指令，这个时候就会触发原来正在下线中的datanode

中止下线的过程

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/062\_再看DataNode下线时候的refreshNodes源码流程/笔记.doc**

重新回过头来看一下，下线一批DataNode节点的时候，这个不是会执行refershNodes的指令吗？看一下这个指令对应的源码的流程。生产环境遇到的问题优化的过程，其实又是跟下线机器有关系的

如果同时下线多台机器，可能会导致有些block数据会永久性的丢失，把一个block所有副本所在的机器都下线了，导致那个block的复制任务生成的时候，选择一个source datanode，就没法选择了

如果找不到source datanode，就找一个decommissioned datanode，还存活，可以工作运行，但是他已经下线了，也可以在最后作为一个兜底性的source datanode来复制一个block副本给其他的datanode

刚刚申请下线一批DataNode，生成了大量的block复制任务，结果突然之间中止DataNode下线，会发现大量的残留的block复制任务还在线上运行，中止DataNode下线之后，但是并没有清理掉之前下线的时候生成的一大堆的block复制任务

如果你要下线一台datannode，一定会导致那个datanode上的大量的bock副本就没了，此时就会导致喝多block的副本数量<3，此时就必须对那些block生产复制任务，去复制对应的副本

保证每个block的副本数量维持在3

最最核心的一点，就是说必须是在下线的时候，要保证这个datanode上的所有的block副本都是放到了复制队列里去，后面就会有一个专门的线程，ReplicationMonitor，他会监听neededReplications队列

对里面所有的block生成复制任务，放到source datanode的内部的复制任务列表

`

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/071\_生产环境问题：DataNode停机几小时之后重启耗时几分钟/笔记.doc**

生产环境遇到的一个问题：有一台DataNode停机了几个小时，做一些集群的运维的操作，深更半夜的在哪儿搞，重启，重启的过程中发现DataNode一遍来说启动就几秒 或者几十秒，但是他那次重启一下子搞了好几分钟，甚至10分钟

DataNode慢启动的问题：我们排查了源码，改造了源码，提升了DataNode长时间停机后重启的效率

万一以后可能还有这种问题，停机几个小时，重启，都很慢的话，其实是非常耽误事儿的，做了源码的改造

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/066\_生产环境问题：中止DataNode下线之后还是有大量block副本任务/笔记.doc**

生产环境：下线了一批DataNode，突然临时有意外，要中止某些DataNode的下线，于是就按照之前说的那个流程执行了操作，无非就是从excluded列表中挪出要中止下线的机器，就可以了

中止下线了之后，hdfs集群里还是积压了大量的复制block任务，那些任务还是在大量的集群里复制，导致集群的压力很大

下线datanode，会生成一大批的复制任务在队列里；中止下线datanode，按理说应该是可以把那些复制任务都取消的，但是没有取消，中止下线了datanode之后，大量的复制任务还是在运行中

中止下线datanode好像没有起到我想要的效果

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/065\_对一个正在下线的DataNode中止下线的源码流程分析/笔记.doc**

正在下线的datanode，中止下线的过程

如果你对正在下线的datanode突然中止下线的话，主要就干一件事情，他就是遍历一下这个datanode所有的block，如果block的副本数量过多的话，此时就会指示某个datanode去删除一个block副本

保证block的副本数量不要成4,5；必须是3

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/072\_DataNode启动流程的源码再次回头梳理分析一下/笔记.doc**

再来看一次DataNode启动的过程，BPOfferService读写锁频繁竞争的优化

BPServiceActor会跟NameNode进行握手通信，会从NameNode获取到一份NamespaceInfo的数据，获取到这份数据之后，就会回调BPOfferService的一个方法，来设置这份NamespaceInfo的数据

我们当时分析那个log，看那个DataNode启动的时候卡住，卡了很长时间，当时就是分析那个log，然后就是看到了这行log：Adding Block Pool 33424。所以在源码里搜索，就知道，原来卡是卡在哪儿了？

DataNode获取到NamespaceInfo之后回调BPOfferService的方法，进行存储的初始化，接着其实会执行BlockPool初始化的东西

FSVolumnImpl，一块磁盘，就是说可能会有多个盘，他一个blockPool可能会分散在多个盘中，所以会有多个FSVolumnImpl的概念

在上面的那行log之后，又看到了一行log：Scanning Block Pool on Volumn ........，就知道肯定是卡在这里了，这里启动的一些线程卡住了，导致DataNode启动流程就卡住了

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/070\_重新复盘一下中止下线后是如何立马清理残留复制任务的/笔记.doc**

复盘一下修改后的源码

下线的时候，放大量的复制任务到needReplications里面去。有些任务已经执行了，有一些block副本已经复制了，就导致block副本数量可能已经过多，那些任务已经不再neededReplications

还有一些副本复制的任务还残留在neededReplications里面，此时还没执行复制任务，但是还残留这些复制任务

避免说中止下线过后，还有大量的复制任务残留在neededReplications里面

3616118202

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/078\_修改hadoop源码之后如何打包成线上可执行软件包/笔记.doc**

持续几周，做了很多源码层面的改进和优化了，起码优化了8个问题，每周都会进行两块源码优化。肯定是你的线上生产集群遇到了一些问题，你看到了一些现象，只有你看到了现象之后，才会来分析源码。

从源码里分析出了问题，才会来优化源码

（1）NameNode突然宕机，日志里打印出来了说无法发送数据到指定数量的JournalNodes集群里去

JournalNodes到底宕机没宕机呢？肯定没宕机的。分析源码，为什么会触发突然NameNode退出呢？看一下jvm gc的日志，发现那个时间刚好出现了fullgc，此时就可以判定出来具体的问题，就是fullgc导致的

优化源码，解决这个问题

（2）某一次在线上生产集群执行下线datanode操作的时候，发现下线多台datanode过后，有一些文件出现了异常，block数据丢失了，检查日志就会发现说，source datanode无法选择

分析源码，就知道是因为当时3个副本所在的datanode都被下线了，就会导致无法选择一台source datanode来复制副本，优化源码，解决这个问题

（3）上传文件的时候，某个文件上传完毕，此时突然几台datanode都故障了（有一个机房尝试搭建了独立集群，专门服务于一个业务，用的是虚拟机，一个宿主机上几台虚拟机），宿主机宕机 ，就导致几台虚拟机一起宕机了。文件里面的block的数据丢失了。

分析源码，就知道上传完毕以后没有进行fsync，就导致机器宕机的时候，数据丢失了，优化源码

（4）某一次线上生产集群，业务方启动了大量的计算任务，导致大量的临时小文件的产生，导致大量的edits log高并发写入，瞬间就是打满了两块缓冲区，导致namenode突然出现一些线上的hang死

通过分析线上的jvm的dump快照，发现大量的线程在等待锁，此时就会发现其实就是缓冲区过小导致的，优化一下源码，调大缓冲区的大小就ok了

（5）上传文件的时候，速度有一点慢，锁的问题，非公平锁的策略，加锁的时间过长，导致线程争用。分析datanode端的上传文件的源码，做出了锁的优化，提升了上传文件的整体的性能。

（6）在执行一些下线datanode运维操作的时候，datanode执行复制指令的速度有点慢，分析了一下源码，注意到执行指令会加写锁，分析读写锁的冲突。优化源码，提升下线datanode运维操作的一些性能。

（7）中止下线一批datanode之后，发现还是积压了大量的复制任务在集群里运行，导致集群资源紧张。分析下线的源码和中止下线的源码，最后优化了源码之后，对中止下线这个行为的效率提升了。

（8）datanode停机几个小时重启过慢，要几分钟甚至10分钟，分析启动的流程里面的源码，就会发现同步执行du命令，导致过慢。优化源码，改一下配置，就可以让以后重启datanode很快。

部署到线上去呢？

（1）写一些单元测试，测试你修改的这块源码：留一个作业，你可以挑选几个单元测试来看看，尤其是什么呢？你修改的那块源码本身就有单元测试，此时你修改源码过后，你可以参考一下你原来修改的那块源码对应的单元测试。自己来写一下。

（2）打包：百度，hadoop如何打包，耗费很多很多的时间，hadoop-2.6.5-zhss-0.1.tar.gz，部署到线上，执行集群的升级操作，升级一下他的版本。集群如何升级，停止，然后替换你最新的版本源码，再次启动。

（3）线上观察一下，原来的问题是否解决了

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/075\_深入分析如何对磁盘执行du命令获取磁盘使用空间数据/笔记.doc**

如果说停机时间超过了10分钟的话，此时就会触发同步执行一个du命令来获取磁盘空间使用的情况，缓存到dfsUsed命令里面去，他后面肯定会有一个DU后台线程定时的不断的去监控这个磁盘空间的使用情况

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/068\_结合源码想一想解决方案：如何中止下线后立即清理残留复制任务/笔记.doc**

我们可以在源码中加一段代码

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/079\_学习课程之后自己未来应该如何尝试对源码进行二次开发/笔记.doc**

大公司里，大数据团队一般来说有专业的分工：

（1）数据平台团队：技术最最强悍的团队，每个人的技术都是很牛的，全公司大数据技术领域最高水准。把你需要的各种大数据的技术，部署好对应的集群，监控、运维和管理。日常主要是大量的读源码，仔细的读源码。源码优化、troubleshooting。

（2）数据应用团队：会围绕基础的平台和集群，开发一系列的大数据工具链平台，ETL开发平台、作业调度平台、数据同步平台、数据采集平台、日志埋点平台、BI可视化平台、数据分析平台、数据治理平台、决策分析支撑平台、用户行为分析平台、实时数据仓库平台、实时分析平台

（3）数据仓库团队：专门基于数据应用团队提供的各种平台，操作那些平台，数据仓库建模，分析数据，写ETL（SQL），产出各种数据应用需要的一些报表，通过hive/spark调度一大堆的作业，跑出来的结果放到mysql，es，kylin

（4）数据挖掘团队：AI，人工智能，机器学习、深度学习，用一些大数据技术，跑一些机器学习的算法，针对公司各种线上的业务，用机器学习产出的结果来支撑线上业务，个性化推荐、智能搜索、智能物流调度

（5）数据开发团队：中小型公司，大公司，分工很细了

如果是中小型公司：数据应用+数据仓库，会合并为一个团队，数据开发工程师，属于万金油，需要数据需求，就是写一些SQL，放到集群里去跑，跑出来的数据填充到一些地方去，mysql，kylin

实时的数据，写一些flink的作业来跑

自己手工写一些java web的界面系统，展示数据出来

（6）大数据开发工程师：小公司，创业公司，大数据团队就几个人，万金油，你可以搭建和部署集群，自己写一些SQL和作业放上去跑，自己写一些web系统提供数据来展现，你还懂一些机器学习的算法，可以spark mllib跑一些智能化一点的任务

你如果要对源码进行二次开发，如果遇到一些问题以后，要学会自己分析日志、分析jvm堆栈、分析jvm gc、分析源码、源码优化，解决问题。多读源码，可能会有一定的灵感，有一块源码可以优化。

包装的特别高大上的一些项目，XXBAT的个性化推荐项目，XX电商用户行为分析的项目，10，20。你刚毕业，或者你刚转型，大数据方向，学结果上述项目，难度不高，可以让你快速进入小公司干大数据

hdfs高级的一些功能，缓存块，短路读，hadoop 3.0提供的纠删码，线上生产环境故障和解决方案，但是不需要修改源码就可以采用一些技术方案来解决线上的问题，提示一下，如果让你从0开始空手搭建一套几十个节点的hdfs集群，运维和管理，整套的思路

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/077\_对DataNode重启间隔参数进行配置化改造避免硬编码/笔记.doc**

改造源码

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/063\_DecommionManager的后台监控线程是如何复制block的/笔记.doc**

DecommissionManager是一个组件，包含了一个后台线程，他的责任就是不断的检查那些处于DECOMMISSION\_IN\_PREOGRESS状态的datanode，督促他尽快的把所有的block副本都要放到复制队列里去

直到所有的副本任务都在复制了，此时就可以把那个datanode转成DECOMMSSIONED状态

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/076\_找到原因：源码中的参数硬编码导致了重新执行了耗时的du操作/笔记.doc**

重启时间超过了10分钟（代码里的硬编码，不可配置），就一定会让loadDfsUsed()返回一个-1的值，也就是不使用之前缓存好的dfsUsed值（磁盘空间使用量），此时就会导致DU线程的构造函数里，会发现拿到了一个-1

此时就会同步执行一个外部的：du -sk /dir，命令，分析当前磁盘空间使用情况，首先执行那个命令就比较慢（在一个数据量特别大的机器上），使用IO流读取数据，分析，等等，这些操作其实可能是比较好使的

DataNode停机几个小时，这个时候机器上的数据肯定是没有变化的，此时重启一旦执行了耗时的du操作之后，就会导致DataNode重启时间过长，几分钟的耗时，10分钟的耗时，DataNode重启慢启动的问题

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/069\_源码二次开发：中止下线后从副本复制任务队列中清理任务/笔记.doc**

源码改写一下

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/074\_再次仔细阅读一下BlockPoolSlice的构造函数的源码/笔记.doc**

读到了BlockPoolSlice的构造函数的源码

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/062~079资料/073\_FsVolumeList的添加blockPool操作的源码仔细走读一下/笔记.doc**

FSVolumnImpl，里面初始化blockPool的过程

如果说你没遇到生产环境的故障，其实没必要对里面所有的源码细节都读的特别的仔细，主要是跟我们之前的源码课一样，把握住一个开源大数据系统，核心的架构、机制、原理和流程就可以了

主要是用，在生产环境里面来用，做日常的运维的操作，监控、运维、管理、源码级的系统优化、源码级的troubleshooting，用的过程中肯定会遇到一些问题：优化类，报错类。DataNode慢启动，就是一个优化类的问题，他没有影响系统故障，但是导致你的系统运行的不太好

锁并发的优化，在集群运行的过程中，你一定会发现一些现象，比如说BPOfferService的读写锁竞争问题，我们是发生一个问题，在执行一些运维操作的时候，比如说有大量的指令下发给datanode

此时会发现datanode内部运行的不太通常，他有一些行为和操作比较慢一些，此时对jvm dump出来一份快照，分析一下，里面到底是怎么回事，有一些线程有很明显的线程的锁wait的操作

通过分析源码，大概就知道了，原来是读写锁频繁的竞争导致了一些指令执行的效率不是太高

报错，DataNode一下子下线了多台机器，直接导致error log日志就出来了，就是说有些block数据丢失了，因为找不到source datanode，这种是属于troubleshooting，线上生产环境有了报错的故障

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/108\_通过源码搜索找找dfsclient何时会申请建立共享内存段/笔记.doc**

requestShortCircuitShm()

首先dfsclient得使用这个方法跟datanode申请建立共享内存段，专门用于维护短路读block的状态的

好像是说，DfsClientShmManager里面好像是使用了一个Sender，这个Sender里有一个out输出流，我们怀疑这个out输出流就是通过unix domain socket用来跟datanode发送请求进行进程间通信的

向datanode发送了一个请求，这个请求就是开辟一个共享内存段

DfsClientShmManager.requestNewShm()，是用来申请共享内存段的一个入口方法

预告，就是说，后面我们会告诉大家说，这个读取数据，走的是DFSInputStream，这个东西里面会通过短路读的方式来读取数据，在这个里面就会触发DfsClientShmManager.allocSlot()方法的调用

ShortCircuitCache.allocShmSlot()

BlockReaderFactory.createShortCircuitReplicaInfo()

ShortCircuitCache.create() -> fetchOrCreate()

BlockReaderFactory.getBlockReaderLocal() -> build() -> BlockReader

BlockReader是什么吗？是DFSInputStream一个关键组件，专门用来跟datanode通信读取block数据的，之前给大家讲解的是普通的BLockReader，他是直接通过tcp连接跟datanode进行网络通信，最后传输block数据的

如果是短路读的话，此时就会通过BlockReaderLocal，使用短路读的方式，直接从本地磁盘文件读取数据，提升性能

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/104\_根据日志以及线上集群问题进行源码级问题定位和修复/笔记.doc**

（1）搞起码几台机器，10多台机器，部署一个hdfs集群（HA高可用架构），实际生产环境建议可以使用cdh企业级大数据平台（开源的）来进行部署、运维和监控

（2）你就可以往这个hdfs集群里存放数据了，也可以从集群里读取数据，集群用起来了

（3）hdfs集群源码级别的架构原理：集群架构、元数据架构、数据写入、数据读取、集群容错，背后的原理，你已经全部都掌握了。他是不会对社区版本做内核级别的过多的修改的，外围的一些企业级功能的开发

（4）随着你用不断的用这个集群，是不是会不断的往hdfs集群里放入数据，数据量是不是会越来越大，是不是hdfs集群里每台机器的磁盘空间的占用量越来越大，可用的剩余空间越来越小，当你的集群的每台机器的可用磁盘空间越来越少的时候，你发现集群的磁盘快满了，此时就需要扩容机器，balancer

（5）创建目录、创建文件、写入数据、读取数据，这要用hdfs就是这么几块，但是在用的过程中，hdfs内核源码可能会有一些问题，导致线上生产集群有一些问题，hdfs二次开发给大家都介绍过了很多我们生产集群遇到的一些问题和如何源码二次开发以及修改的，初步的有了源码级定位问题和解决问题的经验和能力。看到hdfs集群（namenode和datanode）的日志，jvm gc，cdh看看日志的一些定位

（6）

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/113\_dfslicent申请磁盘文件描述符的源码实现剖析（二）/笔记.doc**

dfsclient是如何获取到文件描述符的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/121\_块缓存任务的执行：将磁盘文件中的数据块放入内存缓存/笔记.doc**

datanode接收到块缓存命令之后是如何从磁盘读取出来放入内存缓存的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/101\_对迄今为止学到的hdfs相关课程进行总结/笔记.doc**

3616118202

hdfs源码课，还有二次开发课的100讲，做一个简单的总结

hdfs源码剖析

（1）集群架构

（2）元数据架构

（3）数据写入

（4）数据读取

（5）集群容错

hdfs二次开发和系统优化：源码级分析线上问题，如何通过源码二次开发解决问题

高阶的特性：异构存储架构，内存存储引擎

集群扩容：balancer，集群数据负载均衡

假设，你现在是一个大数据工程师，到了一个初创小公司，我手头有一些数据，我需要把这些数据存储起来，然后后续才可以对这些数据进行分析和计算，从这些数据里提取出来我需要的一些有价值的东西

hdfs集群，大数据架构的课程，是从hdfs集群开始讲

（1）搞起码几台机器，10多台机器，部署一个hdfs集群（HA高可用架构）

（2）你就可以往这个hdfs集群里存放数据了，也可以从集群里读取数据，集群用起来了

（3）hdfs集群源码级别的架构原理：集群架构、元数据架构、数据写入、数据读取、集群容错，背后的原理，你已经全部都掌握了

（4）接下来你就需要去解决他可能遇到的一些问题，比如说集群用着用着，出现一些线上的异常的问题，此时你需要源码级的定位和解决

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/120\_datanode发送心跳的时候是如何接收块缓存任务的/笔记.doc**

datanode发送心跳，是如何接收到块缓存的任务的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/106\_在实践中摸索日常的集群管理与运维的相关操作/笔记.doc**

（1）搞起码几台机器，10多台机器，部署一个hdfs集群（HA高可用架构），实际生产环境建议可以使用cdh企业级大数据平台（开源的）来进行部署、运维和监控

（2）你就可以往这个hdfs集群里存放数据了，也可以从集群里读取数据，集群用起来了

（3）hdfs集群源码级别的架构原理：集群架构、元数据架构、数据写入、数据读取、集群容错，背后的原理，你已经全部都掌握了。他是不会对社区版本做内核级别的过多的修改的，外围的一些企业级功能的开发

（4）随着你用不断的用这个集群，是不是会不断的往hdfs集群里放入数据，数据量是不是会越来越大，是不是hdfs集群里每台机器的磁盘空间的占用量越来越大，可用的剩余空间越来越小，当你的集群的每台机器的可用磁盘空间越来越少的时候，你发现集群的磁盘快满了，此时就需要扩容机器，balancer

（5）创建目录、创建文件、写入数据、读取数据，这要用hdfs就是这么几块，但是在用的过程中，hdfs内核源码可能会有一些问题，导致线上生产集群有一些问题，hdfs二次开发给大家都介绍过了很多我们生产集群遇到的一些问题和如何源码二次开发以及修改的，初步的有了源码级定位问题和解决问题的经验和能力。看到hdfs集群（namenode和datanode）的日志，jvm gc，cdh看看日志的一些定位

（6）对每个技术的高阶特性，hdfs，异构存储架构、内存存储引擎、短路读机制、块缓存机制，你要做到心里有数，很多技术，高阶特性反而用的挺少的，但是你要做到心里有数，关键时刻，万一你真的遇到场景需要使用呢？

集群运维和管理的一些操作，特别的琐碎和细节，集群限流、数据迁移、磁盘性能、集群配置，cdh做起来会方便很多，cdh的时候，一起给大家来讲解和分析，hdfs二次开发和系统优化里，下周，hdfs二次开发这个课最后一周课

源码，两个课程，hdfs核心源码基本都讲完了，代码细节，源码功底托底，你自己后续哪怕自己平时工作做一些集群的运维、监控和管理，都是有巨大的支撑力，遇到问题你再去自己动脑筋去分析，就好做多了

源码级别分析、定位和解决问题，源码二次开发的能力

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/112\_dfslicent申请磁盘文件描述符的源码实现剖析（一）/笔记.doc**

datanode层面是如何获取到磁盘文件的描述符

requestShortCircuitFds()

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/118\_从cacheadmin指令的入口源码开始分析hdfs的缓存机制/笔记.doc**

hdfs cacheadmin，会发送一个缓存请求给namenode，namenode其实他会生成块缓存任务，分配给datanode，datanode发送心跳的时候会收到缓存块的指令，然后把块从磁盘文件里读取出来放到内存里去

分布式计算的时候，无论是短路读，或者是远程网络读，对这个block的读取都可以直接从内存里来读取了

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/114\_基于文件描述符构造短路读副本：ShortCircuitReplica/笔记.doc**

dfsclient已经获取到了磁盘文件的描述符，还有一个共享内存段的slot（短路读副本的状态），构造了ShortCircuitReplica，代表了一个要短路读的block副本

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/117\_通过缓存机制大幅度提升数据仓库SQL的计算性能/笔记.doc**

块缓存机制，异构存储 -> 内存存储，你存一份数据的时候指定存储策略，此时就会固定存放在内存+磁盘里面

你对已经上传到磁盘的数据，可以通过一条指令指示hdfs来把那些数据缓存到内存里去，这样的话后续在计算的时候，计算这些数据的时候，就可以从内存缓存里提取数据来进行计算了

短路读 + 块缓存，大幅度提升你的计算性能

hive sql，两个表join，对一个小表，你就可以使用缓存机制，把小表缓存到内存里去，join的时候大幅度的提升性能了

hdfs cacheadmin -addDirective -path <path>

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/119\_namenode如何根据缓存指令生成datanode的块缓存任务/笔记.doc**

namenode收到缓存指令之后，其实会生成块缓存任务分配给datanode

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/109\_dfsclient申请建立两侧共享内存段的源码剖析（一）/笔记.doc**

先看一下datanode接收到了开辟共享内存段的请求之后是如何处理的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/115\_回头看看DFSInputStream如何自动触发短路读/笔记.doc**

DFSInputStream在构造的时候

基于BlockReaderLocal执行短路读，基于ShortCircuitReplica获取短路读输入流，这个短路读输入流就是基于文件描述符针对本地磁盘文件直接建立的输入流，不是针对datanode建立的输入流

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/103\_如何用企业级的开源大数据平台cdh管理集群/笔记.doc**

cdh，互联网公司，包括各种传统IT公司做大数据，一般来说很多用的都是cloudera hadoop，分成社区版本（apache hadoop），cloudera hadoop，基于社区开源的版本做了二次开发、封装、优化、管理平台

开源的cdh，可视化的方式进行集群的部署、管理、运维、监控，等等诸如此类的一些东西，集成了大量的开源社区版本的大数据技术，hadoop、spark、impala，最后做成一整套大数据平台

开源的，免费来用，但是前提是你的技术足以hold住他，特别是可以看懂他的一些源码，否则如果你hold不住他的话，比如说你用cdh的过程中，遇到了一些问题，不知道为什么对hdfs通过界面做一些运维的操作

结果hdfs集群突然宕机，奇葩的问题，紧张，付费找clouderay公司给他们，说兄弟我需要一些技术支持，support

针对某一个技术领域，比如大数据平台，OLAP分析，可视化分析，BI系统，分布式数据库，推出一套系统，默认是开源的，随便可以用，试用，付钱吧，非常非常昂贵，这种公司，培训，cdh这套系统

国企，传统行业公司，技术人员能力不是特别强，连用都不会用cdh，此时就可以花点钱请人家对你做培训，教会你公司兄弟可以去使用cdh，在用的时候遇到了技术问题，需要找人给你提供技术支持

也会自己研发一些商业级的衍生平台和产品，或者是针对领域、某个行业的整体解决方案，电信行业大数据平台解决方案，金融行业大数据平台解决方案

企业级开发的环境里，你完全可以基于cdh部署hdfs集群，cm来进行集群的监控（资源、性能）、运维、管理、数据的容灾备份、线上问题的监控和诊断，集群扩容，加入新的机器，做balancer

集群运维和管理的操作，在界面上点点点就可以完成了，坑爹的在于，万一出现一些异常的问题，可能会导致集群挂掉，需要你自己精通源码，可以hold住，带着大家把cdh自己本身的一些源码来看一下

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/122\_后续如何通过心跳定期上报自己的缓存块给namenode/笔记.doc**

datanode把块进行缓存之后，后续每隔一段时间会上报自己的缓存块的情况

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/105\_对异构存储、内存存储、短路读、块缓存等高阶特性心中有数/笔记.doc**

（1）搞起码几台机器，10多台机器，部署一个hdfs集群（HA高可用架构），实际生产环境建议可以使用cdh企业级大数据平台（开源的）来进行部署、运维和监控

（2）你就可以往这个hdfs集群里存放数据了，也可以从集群里读取数据，集群用起来了

（3）hdfs集群源码级别的架构原理：集群架构、元数据架构、数据写入、数据读取、集群容错，背后的原理，你已经全部都掌握了。他是不会对社区版本做内核级别的过多的修改的，外围的一些企业级功能的开发

（4）随着你用不断的用这个集群，是不是会不断的往hdfs集群里放入数据，数据量是不是会越来越大，是不是hdfs集群里每台机器的磁盘空间的占用量越来越大，可用的剩余空间越来越小，当你的集群的每台机器的可用磁盘空间越来越少的时候，你发现集群的磁盘快满了，此时就需要扩容机器，balancer

（5）创建目录、创建文件、写入数据、读取数据，这要用hdfs就是这么几块，但是在用的过程中，hdfs内核源码可能会有一些问题，导致线上生产集群有一些问题，hdfs二次开发给大家都介绍过了很多我们生产集群遇到的一些问题和如何源码二次开发以及修改的，初步的有了源码级定位问题和解决问题的经验和能力。看到hdfs集群（namenode和datanode）的日志，jvm gc，cdh看看日志的一些定位

（6）对每个技术的高阶特性，hdfs，异构存储架构、内存存储引擎、短路读机制、块缓存机制，你要做到心里有数，很多技术，高阶特性反而用的挺少的，但是你要做到心里有数，关键时刻，万一你真的遇到场景需要使用呢？

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/111\_仔细盘查短路读源码看何时会申请文件描述符以及构造slot槽位/笔记.doc**

如果要短路读一个block，必须先申请一个共享内存段里的slot槽位

但是先尝试从已有的共享内存段里获取一个slot槽位，但是刚开始如果没有共享内存段的话，此时会跟datanode进行进程间的通信，先分配一个新的共享内存段，并且在两侧都映射好共享内存

然后就可以从共享内存段申请一个slot存放短路读block的状态

他需要对一个短路读的block都需要构建一个ShortCircuitReplicaInfo，就代表了一个短路读副本，就需要一个slot，此时slot已经构造完毕了

文件描述符（代表了要短路读的block磁盘文件在哪里） + 共享内存slot（短路读block的状态的内存映射），ShortCircuitReplicaInfo -> 短路读的block副本

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/116\_再来分析具体如何通过文件描述符直接短路读本地数据/笔记.doc**

BlockReaderLocal已经搞好了，ShortCircuitReplica包含了本地磁盘文件的输入流

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/110\_dfsclient申请建立两侧共享内存段的源码剖析（二）/笔记.doc**

dfsclient获取到了共享内存段的文件描述符之后，是如何将共享内存段映射到自己本地内存的呢？

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/123\_基于内存缓存块进行本地高性能数据读取：零拷贝/笔记.doc**

一旦做了块缓存之后，后续计算的时候读取数据，就可以直接从内存缓存里读取数据了，性能会提高很多

如果短路读，计算任务和数据在一个机器上，实现了短路读 -> 块还进行了内存缓存，此时他就会发起零拷贝的数据读取，进一步提升读取的性能，要经过几个步骤，中间要经过os内核的中转

但是如果使用零拷贝的话，就可以减少os内核的中转转发，提升数据读取的性能

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/107\_提升本地数据读取性能的文件短路读机制实现原理/笔记.doc**

数据本地读

short circuit read

本地读通过datanode的缺点，通过tcp连接读取数据性能开销大，短路读直接绕过datanode从本地磁盘读取数据

linux提供了unix domain socket，进程间socket通信，计算任务通过这种方式跟datanode进行进程间通信，获取本地文件的文件描述符，然后通过文件描述符直接从本地磁盘文件读取数据，绕过datanode

通过下面的方式可以开启短路读的特性，这样当计算任务和datanode在一台机器的时候，就可以自动进行短路读了，可以提升数据读取的性能

<configuration>

<property>

<name>dfs.client.read.shortcircuit</name>

<value>true</value>

</property>

<property>

<name>dfs.domain.socket.path</name>

<value>/var/lib/hadoop-hdfs/dn\_socket</value>

</property>

</configuration>

上面的那个socket路径就是用来设置unix domain socket对应的路径的，然后dfsclient和datanode就通过这个路径的unix domain socket进行进程间通信了，另外还需要打开短路读这个功能

此外datanode会通过一个共享内存段（/dev/shm）来维护短路读block的状态，这个共享内存段里有很多slot，每个slot维护一个短路读block的状态

而且dfsclient和datanode各自都会维护多个共享内存段进行数据映射和同步，dfsclient端是用的DFSClientShmManager管理了多个DFSClientShm，datanode端是用了ShortCircuitRegistry管理了多个RegisteredShm

dfsclient如果要发起短路读，就会通过unix domain socket方式跟datanode进行通信，说自己要短路读某个block，接着datanode就会创建一个共享内存段，在里面维护短路读block的状态

然后datanode在自己这里会创建RegisteredShm映射共享内存段，同时将文件描述符通过unix domain socket传输给dfsclient，dfsclient接收到了之后就会在自己内存里构造DFSClientShm映射共享内存段

接着dfsclient就可以根据共享内存段里映射的文件描述符，直接发起本地数据读取了，可以直接绕过datanode

上述就是短路读的一个核心原理

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/101~123资料/102\_对集群资源使用进行监控以及配合执行集群扩容运维/笔记.doc**

（1）搞起码几台机器，10多台机器，部署一个hdfs集群（HA高可用架构）

（2）你就可以往这个hdfs集群里存放数据了，也可以从集群里读取数据，集群用起来了

（3）hdfs集群源码级别的架构原理：集群架构、元数据架构、数据写入、数据读取、集群容错，背后的原理，你已经全部都掌握了

（4）随着你用不断的用这个集群，是不是会不断的往hdfs集群里放入数据，数据量是不是会越来越大，是不是hdfs集群里每台机器的磁盘空间的占用量越来越大，可用的剩余空间越来越小，当你的集群的每台机器的可用磁盘空间越来越少的时候，你发现集群的磁盘快满了，此时就需要扩容机器

就要往集群里加入一台新的机器，新的机器的磁盘上是空的，没数据的，所以此时必须执行一下balancer，会对集群数据负载均衡，把一些数据挪动到新的机器上去，老机器上删除一些数据，腾出来一些空间了

数据量、磁盘空间监控、机器内存消耗、磁盘IO监控、网络流量监控，等等吧，诸如此类的一些东西，都是需要监控的

hdfs web界面，summary，可以看到集群的整体情况：文件数量，block数量，集群总磁盘空间，使用了多少磁盘空间，剩余可用磁盘空间，线上datanode数量，下线datanode数量，宕机datanode数量

hdfs集群，分布式存储，存放大量的大文件数据，都是放在磁盘上的，监控他的磁盘空间的使用，如果发现磁盘空间满了，必须要做扩容。加入一台机器，balancer，背后的源码如何优化，都给大家讲过了

cloudera hadoop，cdh，开源的企业级大数据平台

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/084\_写入内存存储的block是如何异步写入磁盘文件的/笔记.doc**

LazyWriter不断的从队列里出队，交给了一个组件持久化block的数据到磁盘里去

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/096\_在两个datanode之间是如何完成block数据传输的/笔记.doc**

Dispatch给每个任务都启动了一个线程，来执行block迁移

每个source对应一个Dispatcher线程，每个source里的Task对应一个Move线程，具体执行数据迁移的是Move线程，都是对应着迁移一个block

他会发送一个replace block的请求到target datanode上去

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/099\_通过优化hdfs源码来限制balance不要移动小文件/笔记.doc**

修改Balance里的一些源码，提升一下balance过程的性能

80个小block，10mb；20个大block，128mb

需要迁移1600mb的数据到一个新加入集群的datanode里去，12个大block就可以了；如果你要迁移小block，80 + 5 = 85

改造好了以后，在小文件过多的情况下，会让balance的性能成几倍甚至是几十倍的提升，需要迁移的block数量大大减少了

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/082\_文件的block是如何进入等待写入磁盘文件的队列排队的/笔记.doc**

如果你按照之前讲解的设置了一个文件的存储的策略是内存，datanode会直接把接收到的block给写入到内存里去的，讲一块，主要是介绍为主，不会所有的源码都仔细的来给大家讲解的

会把这个block给放入到一个内存队列里去，后面会有一个组件把这个队列里的数据取出来，再交给另外一个组件，另外一个组件异步的把这个数据刷到磁盘文件里去，LRU算法，在内存满了以后，会检查最近最少使用的block

把最少使用的block从内存里剔除掉

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/091\_如何给线上运行中的hdfs集群动态加入一个节点？/笔记.doc**

随便网上一搜就很简单的

下线节点的时候，在exclude文件里也会放入要下线的datanode，也会执行refreshNodes指令，此时就会触发那个要下线的节点的block复制

slaves加入一个节点，refreshNodes指令就会识别到这个节点就会纳入这个节点到集群里面去，这个datanode一旦启动就会注册到namenode，然后后续就会发送这个心跳汇报自己的所有的情况

balancer是一个非常关键的工具

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/093\_先来看看rebalance指令执行的入口源码流程/笔记.doc**

start-balancer.sh

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/090\_上传文件时申请datanode的时候如何根据存储策略选择/笔记.doc**

datanode会上报自己支持的存储策略，在namenode也是可以感知到的，每个datanode都有多个DatanodeStorage的支持，支持多种存储策略

上传文件的时候，我之前通过storagepolicies指令设置过一个文件的存储策略，是可以支持说根据这个文件的存储策略来选择对应的datanode

内存存储策略那些代码串起来，整个就全部都对应起来了

找到的datanode都是符合你的存储策略的，这个就是一个完整的异构存储结构的实现

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/094\_Balancer组件是如何选择与生成block传输任务的/笔记.doc**

选择从哪些datanode传输哪些block到哪些datanode上去

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/097\_datanode如何接收replace block指令完成数据迁移/笔记.doc**

copy block，复制block，source datanode上面的block是不是也会删除掉，namenode自己就会感知到某个block的副本数量过多了，比如有4个副本了，此时他就会选择block数据过多的那个datanode删除block副本

保持block副本数量只有3个

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/086\_冷热数据隔离：hdfs对异构存储架构的支持/笔记.doc**

异构存储架构，其实就是专门用于将一个hdfs集群里，不同类别的数据存储到不同的介质上去的，比如说那种不常使用的数据就是冷数据，完全可以采用普通的disk策略写到普通磁盘上去

普通磁盘廉价便宜，量还大，存储冷数据很合适，就是读写性能较差罢了

然后如果是那种经常访问的热数据，最核心的热数据，可以跟之前说的那样采用内存存储的策略，直接数据放在内存里，那么分布式计算的时候，性能会非常非常的高，可惜就是内存量太少了，所以只能存少量核心数据

然后就是对于经常访问的一些数据，其实是可以放在SSD上的，也就是固态硬盘，性能比普通磁盘要强十倍百倍，读写速度非常的快

笔记本电脑，128G SSD固态硬盘 + 1T普通磁盘

所以hdfs在hadoop 2.6版本之后就是支持了异构存储的策略，也算是一个相对比较新的功能了，只不过在国内各大公司里用的相对来说不是很多，因为原因主要是一般大数据团队都经过高度分工

数据平台团队会负责管理集群，然后对于那些创建hdfs目录，运行hive/spark SQL作业的数据仓库/数据开发的团队来说，根本不关心hdfs的底层，更加很少很少会手动去设置自己的数据对应hdfs文件的异构存储策略了

而且就我们的实践经验而言，其实大数据里面，大部分都是离线计算作业，对时间开销要求相对较低，慢一点其实还可以接受，只要不是过于的慢，慢到无法接受，那个时候可以考虑深度优化一个作业

但是大部分的情况而言，只要你的集群资源配置合理充足，那么离线计算作业经过一些应用层的参数优化之后，性能相对来说都可以接受，所以正是这些原因，在国内公司里，大数据这块，对hdfs异构存储架构用的其实不多

因此暂时在我们课程里，就是作为一块核心的知识给大家介绍，让大家知道在hdfs集群优化这块，其实有异构存储策略是支持的，但是我们暂时不会带着大家来做这块的实验，大家先知道就行了

当然，原理都理解清楚了，你也完全可以在自己公司测试机器去实验一下这些功能，自己对照官方文档看看，其实操作起来也都不难

异构存储的前提，就是配置一下

<property>

<name>dfs.datanode.data.dir</name>

<value>/home/hadoop/tmp/data,[RAM\_DISK]/mnt/dn-tmpfs,[SSD]file:///grid/dn/ssd0</value>

</property>

其实就是配置当前这个datanode对应的几种不同存储介质挂载的目录，比如有内存的，有SSD的，有普通磁盘的，你都配置好了

然后接着你就是设置某个文件的存储策略就ok了，对于核心热数据，你可以让他存储到SSD或者内存里，普通的冷数据就是写到磁盘里去，不就ok了么，所谓的异构存储，大概就是这个意思

其实就是说每个datanode所在机器，都可以配置异构存储结构，不同的存储介质，内存、磁盘、SSD，等等，还可以是ARCHIVE这种高密度压缩存储的数据结构，读写性能很差，主要用于数据归档，后续很少使用了

这个就是hadoop的异构存储的支持，然后你写数据文件的时候，可以指定每个文件的存储策略，他这个文件的block落地到各个datanode的时候，datanode感知到他的存储策略，就会把block写到对应的存储了去了

如果大家出去买笔记本电脑，电脑运行速度要快，cpu、内存、磁盘，i7处理器，16G + 16G，512G的固态硬盘，玩游戏，开发编程，运行虚拟机，搭建环境，在自己的电脑上任何一种环境都可以搭建了

128g SSD + 1T机械硬盘

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/085\_内存使用量达到极限之后如何用LRU算法清理不常用数据/笔记.doc**

evictBlocks

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/089\_上报后的存储策略是如何在namenode进行保存的/笔记.doc**

datanode通过心跳的时候带上StorageReport，每个盘的存储类型以及使用情况

namenode是如何组织和管理每个datanode支持的多种存储策略的呢？每个DataNode对应多个DatanodeStorage

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/088\_datanode如何通过心跳上报自己支持哪些存储策略呢/笔记.doc**

datanode心跳的时候会上报自己支持的存储策略

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/083\_在队列中排队的内存存储block是如何被异步取出来的/笔记.doc**

LazyWriter，会不断的从队列里取出来需要持久化到磁盘的内存存储的block

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/081\_hadoop hdfs对内存存储的支持以及实现原理/笔记.doc**

首先需要配置linux的虚拟内存盘的配置，这个虚拟内存盘就是基于内存来存储文件数据的意思，必须先配置这个，hdfs才能支持内存存储

sudo mount -t tmpfs -o size=16g tmpfs /mnt/dn-tmpfs/

上面这行命令，就是把tmpfs文件系统挂载到/mnt/dn-tmpfs上去，大小是16g。默认tmpfs文件系统是挂载到/dev/shm的，大小是32g，通过上面的命令，我们就可以挂载到自己指定的目录里去

接着需要在hdfs配置中挂载好的虚拟内存盘

<property>

<name>dfs.datanode.data.dir</name>

<value>/home/hadoop/tmp/data,[RAM\_DISK]/mnt/dn-tmpfs</value>

</property>

最后，就是设置某个文件是要用内存存储策略的

hdfs storagepolicies -setStoragePolicy -path <path> -policy LAZY\_PERSIST

内存的大小是有限的，比如就16g，你如果要是不停的往内存里存放文件block的数据，肯定是吃不消，一个block都128mb，1g内存也就放10个block，一台datanode上有16g内存，也就100多个block

内存放满了以后，就会基于LRU（最近最少使用的策略），他会把最近最少使用的block从内存里剔除掉，容纳后续的其他的block再写入到内存里去

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/080\_分布式计算如何实现100倍性能提升：内存存储引擎/笔记.doc**

分布式计算的性能问题

分布式存储 + 分布式计算的关系，分布式计算的前提是分布式存储，分布式存储 + 分布式计算，hadoop，在hadoop 1.x的版本里，hdfs + mapreduce，hive

数据量很大，分布式存储，就把一大块数据分布式存储在多台机器上，多副本冗余，任何一台机器宕机，保证数据不会丢失

mapreduce，JobTracker、TaskTracker，hadoop 2.x版本里，就是YARN做资源管理和调度

分布式计算，涉及到大量的磁盘读写 + 机器之间的网络传输

很多时候，你会发现比如说你用hadoop生态里的hive，你跑一些大的SQL去执行计算，会很慢，其实导致大数据计算的性能不是特别高的一个最大的瓶颈，磁盘文件的读写，数据在hdfs管理的时候，都是放在磁盘上的

分布式计算的时候，必然涉及到大量的磁盘文件的读，就是很慢的过程

如果要100倍的提升分布式计算的性能，应该如何做呢？就是在各个机器上把数据放在内存里，不要放在磁盘文件里，分布式计算的时候，读取数据就直接从内存里读取了，这个就很厉害了

避免大量的磁盘文件的读写，大幅度的提升分布式计算的性能

现实很残酷的，还很难做到真的大数据存储在内存里，内存资源是很宝贵的，物理机，64G，2T，你如果基于磁盘存储数据可以放2T，基于内存来放数据，可能也就二三十G，放不了大数据了

hdfs是支持内存存储引擎的，把数据放在内存里的，在工业界很少用，不是特别的成熟，落地的案例较少，使用场景很有限制

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/100\_对rebalance指令优化带宽参数以及定向迁移数据提升性能/笔记.doc**

hdfs dfsadmin -setBandwidth 100m

默认是10mb

./start-balancer.sh -include f hostfile，你可以自己手动指定只能从哪些source datanode来迁移数据，可以更好的选择一些数据量过多的datanode来进行迁移

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/087\_先看看hdfs对异构存储架构的核心实现原理是什么/笔记.doc**

分析了一下异构存储的架构实现原理

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/098\_hadoop的balance工具移动大量小文件的性能低下问题/笔记.doc**

如果有大量的小文件

比如说你现在是有一个datanode，上面有100个block，80个block都是小文件，每个block可能就10mb，800mb，另外20个是正常的大文件，每个block都是128mb，2500mb

另外一个datanode是新加入的，1600mb左右的数据，此时可能就会变成80个block对应的小文件都需要进行迁移，但是此时才800mb，还需要迁移5个128mb的正常大小的block，80个小block + 5个正常block

source datanode要迁移85个block到target datanode上去

通过源码都看的很清楚了，迁移一个block的成本是很高的，一个Move线程是对应了一个block的迁移，每个block的迁移过程很复杂的，大量的网络通信，资源开销，传输数据，释放资源

如果小文件过多的话，就会导致balancer的过程很慢，性能很差

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/095\_block传输任务是如何交给Dispatcher组件进行执行的/笔记.doc**

source，Task<target, size>，Task<target, size>，Task<target, size>

Dispatcher会给每个数据迁移的任务，都启动一个线程，这个线程决定把哪个block传输到target datanode上去

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/080~100资料/092\_rebalance指令将文件block动态平衡重分配的原理/笔记.doc**

为什么在动态加入了一个节点之后，start-balancer.sh

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/053\_在复杂的DataNode启动流程中注意到NamespaceInfo数据/笔记.doc**

在分析完毕了datanode启动的过程，因为我们读hdfs源码积累大量的基础，所以再次看源码的时候，速度是相当的快，快如闪电。我们来注意一个关键的数据，NamespaceInfo。

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/044\_非公平锁策略可能会导致FSDatasetImpl中的锁有什么问题？/笔记.doc**

非公平锁，适合的场景，就是说每个线程持有锁的时间比较短，大家呢就，就是可以非公平竞争抢占，哪怕是线程2等待了一段时间，前面别好几个别的线程插队获取了锁，插队的线程获取到了锁之后，可能就是几微妙就直接释放锁了

对于我们的hdfs的FSDatasetImpl里的锁，用非公平锁就不太合适了，有线程在那儿队列里等待，前面好多线程过来插队加锁，每个线程占用锁的时间很长，因为在执行耗时的文件操作，几十毫秒，几百毫秒，甚至是几秒

非常不合理的导致有的线程先来，结果楞是等待了好几秒才加到了锁

对于每个线程持有锁的时间过长的情况，是不太合适用非公平锁，不太合适了

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/055\_BPOfferService中的getBlockPoolId为何是个高频率读操作？/笔记.doc**

以下是施加写锁的场景：

（1）BPServiceActor获取到了NamespaceInfo，设置这个数据

（2）BPServiceActor准备向NameNode注册的时候，会创建一个DatanodeRegistration数据

（3）BPServiceActor注册成功了之后，会来设置DatanodeRegistration这个数据

（4）如果DataNode关闭，此时就会关闭某个BPServiceActor，此时会清空一些数据

（5）每隔3秒钟跟NameNode进行一次心跳，如果发现NameNode的active/standby状态发生了变化，此时就会更新一些数据

（6）每次发送心跳给NameNode都有可能会带回来一些指令，比如说通知DataNode复制某个block副本到其他的DataNode上去

以下是施加读锁的场景：

（1）getBlockPoolId()，从BPOfferService中管理的一个NamespaceInfo数据里提取出来对应的NameNode的blockPoolID这个核心的id数据

（2）getNamespaceInfo()，直接获取核心的NamespaceInfo数据

（3）获取Active NameNode对应的rpc接口代理

证明一个结论：getBlockPoolId()，这个操作，绝对是DataNode里的一个高频率的很频繁的操作，证明了高频率的通过执行这个操作来加BPOfferService里的读锁

哪些地方是高频率的调用BPOfferService里的getBlockPoolId()：

（1）每隔几分钟的频率去汇报增量更新的block给namenode

（2）blockReport()全量块汇报

（3）cacheReport()缓存块汇报

（4）每隔3秒钟都会发送一次心跳，都会调用这个操作

（5）其他的一些操作

尤其是每隔3秒钟发送心跳，以及每隔几分钟汇报block给namenode，这些东西，其实都是高频率的操作，所以说BPOfferService里面的getBlockPoolId()操作是高频率的操作，证明了会高频率的加读锁

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/045\_采用公平锁策略之后又是如何优化FSDatasetImpl加锁效率的？/笔记.doc**

如果用公平锁的策略如何优化这种加锁耗时较长的锁性能

每个线程加锁时间耗时较长的场景中，其实用公平锁策略是比较合适的，就可以让每个线程按照队列排队先来后到的顺序，大家都公平的有加锁的机会。先来的线程先排队，先有机会去加锁。

哪怕每个线程关耗时时间较长，也没关系，大家都是排队，一个一个的来加锁

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/048\_回顾一下重要的锁优化策略：尽量减少锁占用的时间/笔记.doc**

如果大家关注过咱们java并发的课程的话，锁优化策略，其实说起来非常的简单，但是核心在于，我们在那个并发课程里全程实战，大量的项目和案例驱动，把并发的技术给讲清楚，并发优化实践这块

尽量减少线程对锁占用的时间开销，一个线程加锁之后占用锁要10毫秒；另外一个线程加锁之后占用锁要10秒；前者的话，可以保证10毫秒之后让其他线程进来加锁和执行代码；后者的话，会导致大量的后面线程会等待10秒，或者是几十秒后才能获取到锁

若果一个线程在等待锁的话，是什么都干不了的，一个机器上可以开启的线程资源的数量是有限的，如果你锁并发性能太差的话，会导致机器上宝贵的线程资源多hang在那儿等待锁的获取

尽最大努力来减少一个线程加锁之后占用锁的时间的开销，保证一个线程快速的加锁和释放锁，然后其他的线程可以快速的加锁和释放锁，提升系统整体的吞吐量和并发能力

最最经典的一个锁优化的案例，edits log机制，双缓冲机制，一个线程在刷数据到磁盘的时候，不占用锁的，保证其他的线程可以快速的加锁写数据到内存缓冲里，大幅度的提升了多线程并发写edti slog的性能

在锁代码块仅仅是操作一下文件，但是还没执行fsync，这个性能要提升很多了，大幅度的降低一个线程对锁时间的占用

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/047\_在ReentrantLock加锁代码中执行耗时的fsync操作有问题吗？/笔记.doc**

当然有问题了，把那个锁给优化了一下，公平锁的策略，每个线程加锁之后，如果执行完一大堆的文件操作，然后还要在锁代码块里来执行fsync操作的话，fsync操作是把数据直接强制刷到磁盘上去的

fsync是很耗时时间，上秒级别的都有可能

很有可能给就会导致每个线程加锁之后，执行一大堆的文件操作，而且直接执行fsync操作，会导致锁占用的时间会特别的长

很多个线程过来的话， 大量的线程会在那里排队，一个线程接一个线程的执行文件操作和fysnc操作

fsync放入到锁代码块里去，直接会导致你的在这块文件锁的并发性能会很差

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/061\_复盘一下源码重构之后是如何减少BPOfferService读写锁争用的/笔记.doc**

以下是施加写锁的场景：

（1）BPServiceActor获取到了NamespaceInfo，设置这个数据

（2）BPServiceActor准备向NameNode注册的时候，会创建一个DatanodeRegistration数据

（3）BPServiceActor注册成功了之后，会来设置DatanodeRegistration这个数据

（4）如果DataNode关闭，此时就会关闭某个BPServiceActor，此时会清空一些数据

（5）每隔3秒钟跟NameNode进行一次心跳，如果发现NameNode的active/standby状态发生了变化，此时就会更新一些数据

（6）每次发送心跳给NameNode都有可能会带回来一些指令，比如说通知DataNode复制某个block副本到其他的DataNode上去

以下是施加读锁的场景：

（1）getBlockPoolId()，从BPOfferService中管理的一个NamespaceInfo数据里提取出来对应的NameNode的blockPoolID这个核心的id数据

（2）getNamespaceInfo()，直接获取核心的NamespaceInfo数据

（3）获取Active NameNode对应的rpc接口代理

证明一个结论：getBlockPoolId()，这个操作，绝对是DataNode里的一个高频率的很频繁的操作，证明了高频率的通过执行这个操作来加BPOfferService里的读锁

哪些地方是高频率的调用BPOfferService里的getBlockPoolId()：

（1）每隔几分钟的频率去汇报增量更新的block给namenode

（2）blockReport()全量块汇报

（3）cacheReport()缓存块汇报

（4）每隔3秒钟都会发送一次心跳，都会调用这个操作

（5）其他的一些操作

尤其是每隔3秒钟发送心跳，以及每隔几分钟汇报block给namenode，这些东西，其实都是高频率的操作，所以说BPOfferService里面的getBlockPoolId()操作是高频率的操作，证明了会高频率的加读锁

证明一个结论：写锁也是高频的加的，会导致高频率的读写锁的竞争

（1）分析一下每次发送心跳给NameNode可能会带回来一些指令

（2）如果需要执行指令的话，那么就需要加写锁，哪些场景下会执行指令：复制块、删除块、缓存块、finalize块、恢复块

（3）要么不执行，要么就是短时间内可能就是高频率的执行指令，比如说你现在突然之间集群里就是下线了几台datanode机器，此时他们那些机器上的block副本，都需要下发指令给其他的datanode进行复制

（4）你在执行一些运维操作的时候，短时间内会有大量的块操作的指令，此时就会在一些运维操作的，短时间内频繁的加写锁

在某些场景下，短时间内会频繁的加写锁，读锁本来就是每次发送心跳都会加读锁，此时可能会在一段短时间内出现读写锁的频繁的冲突

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/049\_重构finalizeReplica操作的源码让其返回目标文件对象/笔记.doc**

finalizeReplica操作现初步完成了重构

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/052\_重新回顾一下DataNode启动时的注册机制的流程/笔记.doc**

回顾一下DataNode启动的时候会干哪些事儿，有一个关键的事儿就是往NameNode去注册自己，注册的这个过程我们来着重的关注一下。从这个点切入，我们后续要优化这个相关的一些源码

读写锁高并发冲突的问题

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/046\_回顾一下之前在finalizeBlock操作中加入的fsync机制/笔记.doc**

回顾一下之前为什么要在finalizeBlock操作中加入fsync机制

就是要对finalizeBlock中的fysnc机制对锁的并发性能的影响来分析一下，以及做出优化

如果你不加fsync操作，假如说我们仅仅是在block上传完毕之后，就是将数据flush到os buffer，内存，此时DataNode所在的机器宕机了，会导致os buffer（内存）中的数据会丢失，对不对

fsync，在上传完毕一个block了之后，尽快的立马将os buffer中的数据fsync强制刷到磁盘文件里去，哪怕此时DataNode所在的机器宕机了，也不要紧，机器重启之后，数据已经落地到磁盘文件里去了

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/054\_仔细分析一下BPOfferService中的高频出现的读写锁机制/笔记.doc**

BPOfferService中有一个核心的数据，NamespaceInfo，他是在BPServiceActor通信之后获取到了会给BPOfferService。BPOfferService里的NamespaceInfo核心数据，是被执行各种读写操作的时候，其实是被施加了大量的读写锁的

以下是施加写锁的场景：

（1）BPServiceActor获取到了NamespaceInfo，设置这个数据

（2）BPServiceActor准备向NameNode注册的时候，会创建一个DatanodeRegistration数据

（3）BPServiceActor注册成功了之后，会来设置DatanodeRegistration这个数据

（4）如果DataNode关闭，此时就会关闭某个BPServiceActor，此时会清空一些数据

（5）每隔3秒钟跟NameNode进行一次心跳，如果发现NameNode的active/standby状态发生了变化，此时就会更新一些数据

（6）每次发送心跳给NameNode都有可能会带回来一些指令，比如说通知DataNode复制某个block副本到其他的DataNode上去

以下是施加读锁的场景：

（1）getBlockPoolId()，从BPOfferService中管理的一个NamespaceInfo数据里提取出来对应的NameNode的blockPoolID这个核心的id数据

（2）getNamespaceInfo()，直接获取核心的NamespaceInfo数据

（3）获取Active NameNode对应的rpc接口代理

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/050\_重构finalizeBlock操作的源码将fsync操作挪到加锁代码之外/笔记.doc**

挪动fsync操作到锁代码块之外

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/058\_基于volatile机制将注册之后得到的blockPoolId缓存起来/笔记.doc**

volatile来缓存一下blockPoolId，多线程的可见性，必须回头去看之前讲解的java并发课，讲解了volatile底层内存屏障和硬件级别的实现原理，到底是如何保证了可见性的问题的，对不对

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/042\_使用ReentrantLock对FSDatasetImpl中的文件操作加锁/笔记.doc**

FSDatasetImpl中的锁优化的一套东西，源码修改，优化锁并发的性能

在hadoop 2.8.x版本以后，开始在FSDatasetImpl中加入了一个锁，专门对一些文件操作进行加锁。如果你多个线程并发的对一个文件进行修改和写入的操作，可能是会有问题的。所以在hadoop里面，在比较新的版本

就是把这个锁给他加上，后续我们再来一点点分析这个锁的并发性能如何优化

DataNode突然所在的机器宕机的话，可能会导致刚刚上传完毕的block会丢失，他上传完一个block之后没有及时的进行fsync的操作，仅仅是写入os buffer就结束了，此时如果机器宕机的话，会导致这个文件的数据会丢失的

FSDatasetImpl.finalizeBlock()方法里 => finalizeReplica()方法，执行了fsync操作，Java NIO里的FileChannel API进行的fsync操作。

finalizeBlock()方法，实际上就是一个非常主要的，要对对文件进行相关的写操作的方法，在hadoop比较新的版本里面，就加入了这个patch，优化。凡事涉及到对一个磁盘文件可能存在多个线程并发写入的情况

尽可能需要加锁的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/057\_DataNode启动过程中在什么地方可以获取到blockPoolId呢？/笔记.doc**

分析清楚了一个点，写锁在某些块操作下发大量指令给datanode的时候，会频繁的加写锁，而且有可能加写锁占用的时间还是比较高的，因为他要去触发执行一些块的复制操作，有一些业务逻辑在里面

读锁本身就是频繁加的，getBlockPoolId()会频繁的调用，但是他频繁的加了读锁

getBlockPoolId()操作，不要频繁的加读锁？如果他不频繁的加读锁，不就可以避免某些场景下严重的读写锁的争用问题了吗

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/043\_在Java中非公平锁与公平锁的区别到底是什么？/笔记.doc**

ReentrantLock，公平锁与非公平锁，两种锁策略，默认是非公平锁策略

讲一下非公平锁和公平锁的区别，java并发编程课程里面，讲过ReentrantLock底层的源码，AQS的源码，公平锁和非公平锁策略

石杉的架构笔记，发过一篇文章，也是讲公平锁和非公平锁

《CAS操作，他其实java并发底层的基础》

《大白话聊聊Java并发面试问题之谈谈你对AQS的理解？》

《大白话聊聊Java并发面试问题之公平锁与非公平锁是啥？》

非公平锁，你尝试加锁的时候，很可能是你先过来尝试加锁入队等待，结果后面有一个比你晚来很久的线程，比你先加锁，因为大家都是在竞争吗，锁队列排队，其实只是一个机制，不保证所有的线程都是排队等待

公平锁的策略，他的意思就是说，任何线程尝试加锁的时候，必须判断一下等待队列里有没有人在等待加锁，如果有，那么他必须入队，排队，按照顺序来加锁

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/051\_复盘一下源码重构之后的finalizeBlock整体机制和流程/笔记.doc**

（1）对文件执行大量复制操作的锁是要加的

（2）公平锁的策略，每个线程过来依次排队加锁，避免某个线程长时间等待加锁，被很多线程插队在前面来加锁

（3）fsync机制从加锁代码块中挪出来，降低每个线程加锁后占用锁的时间的开销，fsync性能开销太大了，耗时太多，每个线程加锁之后降低占用锁的时间，尽可能提升多线程并发写磁盘文件的效率

（4）在锁代码块之外，来执行fsync操作，耗时，但是不会长时间占用锁

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/060\_回顾一下重要的锁优化策略：尽量减少线程之间对锁的争用/笔记.doc**

java并发课，锁优化策略，有非常关键的一点，其实就是对锁的竞争尽量减少争用，不要让大量的线程频繁的去竞争一把锁，案例，是微服务注册中心里，实现了一套多级缓存的架构，通过那套架构

尽最大可能去避免了读写锁的竞争

频繁的读写锁的争用，频繁的加读锁， 频繁的加写锁，此时你最好是优化一下这块的代码减少锁的争用的问题

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/059\_优化getBlockPoolId使用缓存好的BlockPoolId避免读写锁竞争/笔记.doc**

核心的源码优化，尽可能在getBlockPoolId()操作里不要加读锁

高频的getBlockPoolId()操作，几乎不会在运行期间加读锁，对写锁的时候，不会出现频繁的争用，低频率的读写锁的施加操作，那其实是无所谓的

# **File Path: /Users/xiaotingting/Downloads/学习资料/02\_hadoop源码二次开发、系统优化以及问题修复/042~061资料/056\_全局角度看getBlockPoolId是如何导致频繁读写锁竞争的？/笔记.doc**

以下是施加写锁的场景：

（1）BPServiceActor获取到了NamespaceInfo，设置这个数据

（2）BPServiceActor准备向NameNode注册的时候，会创建一个DatanodeRegistration数据

（3）BPServiceActor注册成功了之后，会来设置DatanodeRegistration这个数据

（4）如果DataNode关闭，此时就会关闭某个BPServiceActor，此时会清空一些数据

（5）每隔3秒钟跟NameNode进行一次心跳，如果发现NameNode的active/standby状态发生了变化，此时就会更新一些数据

（6）每次发送心跳给NameNode都有可能会带回来一些指令，比如说通知DataNode复制某个block副本到其他的DataNode上去

以下是施加读锁的场景：

（1）getBlockPoolId()，从BPOfferService中管理的一个NamespaceInfo数据里提取出来对应的NameNode的blockPoolID这个核心的id数据

（2）getNamespaceInfo()，直接获取核心的NamespaceInfo数据

（3）获取Active NameNode对应的rpc接口代理

证明一个结论：getBlockPoolId()，这个操作，绝对是DataNode里的一个高频率的很频繁的操作，证明了高频率的通过执行这个操作来加BPOfferService里的读锁

哪些地方是高频率的调用BPOfferService里的getBlockPoolId()：

（1）每隔几分钟的频率去汇报增量更新的block给namenode

（2）blockReport()全量块汇报

（3）cacheReport()缓存块汇报

（4）每隔3秒钟都会发送一次心跳，都会调用这个操作

（5）其他的一些操作

尤其是每隔3秒钟发送心跳，以及每隔几分钟汇报block给namenode，这些东西，其实都是高频率的操作，所以说BPOfferService里面的getBlockPoolId()操作是高频率的操作，证明了会高频率的加读锁

证明一个结论：写锁也是高频的加的，会导致高频率的读写锁的竞争

（1）分析一下每次发送心跳给NameNode可能会带回来一些指令

（2）如果需要执行指令的话，那么就需要加写锁，哪些场景下会执行指令：复制块、删除块、缓存块、finalize块、恢复块

（3）要么不执行，要么就是短时间内可能就是高频率的执行指令，比如说你现在突然之间集群里就是下线了几台datanode机器，此时他们那些机器上的block副本，都需要下发指令给其他的datanode进行复制

（4）你在执行一些运维操作的时候，短时间内会有大量的块操作的指令，此时就会在一些运维操作的，短时间内频繁的加写锁

在某些场景下，短时间内会频繁的加写锁，读锁本来就是每次发送心跳都会加读锁，此时可能会在一段短时间内出现读写锁的频繁的冲突