REHDFS: A random read/write enhanced HDFS

翻译：孙亚博

时间：2018年8月27日

# 摘要

Hadoop文件系统（HDFS）（Shvachko et al., 2010）是一个可扩展，高容错的分布式文件系统，并且可以部署在廉价的硬件设施上。存储在HDFS上的内容是分块存储并且在多个数据节点（DataNodes）上有副本。通过不同的块存放机制使其具有高容错并且可以提高吞吐量以及访问时间。HDFS允许用户：

（1）对于一个要分割的文件自定义其块大小

（2）仅仅执行顺序的读以及追加写操作。HDFS不允许用户执行随机读以及随机写操作。

本文提出一种增强的HDFS（REHDFS）具有以下改进：

（1）提出不同的块放置以及块读取机制

（2）实现随机读写操作

本文的结构已经实现并且得到了验证。提出的**load based**访问机制相对于其他的块检索机制有更好的性能。随机写的特性已经实现并且得到验证。提出了使用悲观以及乐观的模型来验证随机写的操作，通过实验已经实现。

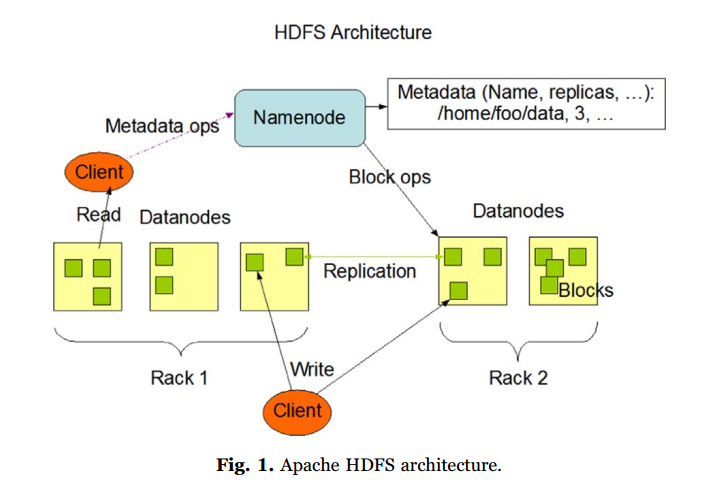
# 简介

全球的数据正以一个快速的速度增长，并且预计在下个世纪将会达到35zb（Meng and Li, 2015）。大量的数据通过社交网络、软件等产生，例如facebook、Twitter、WhatsApp，各种各样的传感器也在不断产生数据。

这些大量产生的数据包含了新兴的市场趋势。大多数的机构如果可以利用这些新兴趋势的信息就可以做出准确的决定。传统的企业使用数据挖掘的工具来分析历史结构化的数据，而现在都在调研可用的工具来分析非结构化的数据以及急速增长的数据来了解市场最新趋势，进一步预测消费者可能的行为。通过这些工具获取的知识正用于提高企业的市场决策，或者说微调他们当前的进程，并且可以完善他们的方案从而可以提高自身的市场竞争力。因此，急需用于处理大数据并且具备可扩展、容错性的分析工具。

过去的几年中，大数据的处理变得常见，不同的方法像Hadoop以及Spark被广泛用于操作以及处理大数据。Hadoop提供了一个可扩展以及容错性好的环境，不仅支持分布式的数据处理，而且支持分布式存储。Spark是一个集群计算框架，通过基于内存的集群计算技术来获得更快的处理速度。Hadoop以及Spark正成为未来数据密集型产品的平台。Hadoop提供了一个数据处理的模块叫做MapReduce。MapReduce允许开发人员使用廉价的机器来开发可以进行并行计算的稳定的应用。传统的基于MapReduce的算法包括俩个阶段。第一个阶段叫做Map Phase，第二个阶段叫做Reduce Phase。Map Phase将键值对作为输入，然后预期产生一系列的中间键值对。Reduce Phase主要聚集不同的Mapper产生的输出。在一个MapReduce任务执行过程中，Hadoop集群为二者分配正确的server。通常，一个大的数据文件被切分为多个chunk，Mapper运行在一个或多个server上，并且每一个server处理一个或多个chunk。Mapper的结果会被排序并且shuffle然后直接交给一个或多个Reducee。Hadoop集群最终从Reducers收集结果然后返回给初始化任务的客户端。

Hadoop集群包括Hadoop分布式文件系统（HDFS）来提供大数据应用中文件的存储。HDFS是一个分布式的文件系统，是可扩展以及高容错的。其最初的目标就是可以在廉价的硬件环境中来搭建。市场中有很多的分布式文件系统，这些分布式文件系统在很多方面不同，包括文件存储策略，容错级别，性能。数据变化，以及对并发的支持等。HDFS与已经存在的分布式文件系统有很多的相似之处。但是与其他分布式文件系统的不同之处也是显而易见的。在Hadoop中，逻辑处理是像数据靠拢，而不是数据像逻辑处理靠拢。这个模式使得HDFS可以提供高吞吐量以及快速访问应用数据并且很适合大数据集的操作。HDFS放宽了一些POSIX的要求来满足文件系统数据的流式访问。HDFS的架构如Fig.1：



HDFS的架构是基于master/slave架构。一个Hadoop集群包括一个NameNode，一个或多个DataNode。Namenode用于管理元数据，包括：构建文件系统命名空间；分配存储文件内容的节点；增强访问控制。Namenode也提供了打开、关闭、重命名以及删除文件或者文件目录的功能，并且分配一个特定的datanode来存放特定的文件块。文件内容划分为一个或多个块。一个块存放在多个DataNode上来实现容错性。一个DataNode管理一个或多个块。DataNode提供HDFS客户端对于块的读写请求。DataNode也满足NameNode对于创建、删除以及备份块的请求。典型的Hadoop部署使用俩个或多个机架来防止出现错误。HDFS不支持对放置在HDFS文件的随机读写操作。

HDFS允许用户对于其上的文件执行读以及追加写的操作。即使是想要修改一个字节的文件，必须创建一个新文件然后使用新文件将旧的文件替换。考虑如下场景，一个项目团队需要去管理与项目有关的一系列文件，文件需要满足高可用以及高容错性，并且这些文件可以被项目中的任何成员进行编辑，因此这些文件需要具备高度的一致性。另外这些文件非常大，而且项目成员会经常对其进行编辑操作，一些项目成员使用不同的工具可以有效的对这些文件进行相关操作。这些工具在项目中被设计使用标准的I/O操作比如read(),write(),seek()等等来操作文件。随机访问以及随机写在这个应用场景中是必须的。支持想这样场景的应用必须保证数据的高可用以及容错性。HDFS的副本机制存放在不同的块，并且每一个块又存放在不同的数据节点上，因此HDFS是一个比较好的选择来存放这样的数据。

一组项目成员或者一系列的操作需要不断的对一系列文件进行操作需要将文件保存在HDFS上。当数据的一致性是一个最重要的因素时，现在将HDFS上的数据拷贝到本地然后进行修改，最后替换HDFS上的文件的方案就不再合适。对于这样的模式，一个开发人员做的修改可能会被另外一个开发人员的修改覆盖。

类似HBase以及其他一些解决方案要求数据是结构化的格式（HBase表是一个键值对的map）。他们在行级别提供了一致性，而在整个文件的一致性上都不能保证。

在HDFS上下文提供随机写操作是一个很难解决的问题，包括一下问题需要解决：

1. 一致的将所有的变化传播到所有的副本上（存储在不同数据节点）
2. 协调不同的节点比如NameNode和一系列的DataNode
3. 增强的数据一致性
4. 需要提供一个回复机制当一个DataNode在执行过程中down掉后

本文提出一个扩展的HDFS（REHDFS），实现以下功能：

1. 允许用户实现不同的块存放以及访问策略
2. 对于给定的文件进行随机访问以及随机写
3. 不同级别的locking 粒度

使用REHDFS，一个应用可以实现对于放置在HDFS上的文件实现随机读写操作。这篇文章的后续按照以下安排：第二章分析问题并且说明相关的工作；第三章说明REHDFS架构以及支持REHDFS特性的算法；第四章实现的细节以及实验结果；第五章实验结论

# 遇到问题及相关工作

这个章节首先分析了问题并且说明了需要在REHDFS中解决的一些子问题，最后是一个相关工作的介绍。

## 2.1 案例场景以及问题分析

为了能够达到容错，每一个存放在HDFS上的文件会被拆分为多个block，并且每一个block会被存放在多个节点上。在实现HDFS架构时候，可以采用不同的块存放策略。HDFS使用默认的块大小来存放文件，并且允许用户来修改。HDFS不支持随机读写操作。

块存放策略（Block placement strategies）：不同的块存放策略导致了不同的容错级别以及访问的速度。简单的副本机制比如将整个文件放在每个DataNode，将每个文件块存放在不同的DataNode需要被评估的。

选择块大小：HDFS对于所有的文件使用默认的块大小（一般是64M或者128M）,并且允许用户在HDFS上创建文件的过程中指定块的大小。块大小的选取对访问性能的影响需要进一步的研究。

Block 缓冲以及检索机制：HDFS客户端能够扩展到cache block来提高性能，客户端从DataNode检索的block会被保存在cache中，后面使用时候可以直接从cache中快速的获得。更进一步的性能优化是可以尝试采用将检索的block不是放在cache中。而是自动触发一个异步的下一个block检索来提高之后的访问速度。

随机读：用户可能想要直接的访问文件特定的部分通过调用**fseek（）**函数。对于访问请求位置的过程需要计算block位置，需要验证存放block的DataNode，最后需要从其中一个DataNode下载block。

随机写：用户可能需要修改文件的特定部分通过**fwrite（）**函数。同样的，与写相关的block需要验证、下载然后修改。修改后所有的block需要追踪然后保存。每一个被修改的副本都需要更新。由大量客户端同步请求的必须保证正确的同步，而且必须解决可能产生的任意冲突。

## 2.2 相关工作

本文所做的工作是与Hadoop生态环境中管理分布式文件相关。分布式文件系统已经投入实践很长时间。并发的访问分布式资源一般是由悲观模式（锁机制）或者乐观模式来控制。从一系列DataNodes选择最佳的DataNode来下载block类似于解决一个平衡问题。本章节尽可能总结其他研究人员做出的重要贡献，这些贡献与本文工作有很大相关性。

Sun的微小网络文件系统（NFS）（Sandberg等1988）被广泛的应用在学术以及工业环境中。NFS提供位置透明以及访问透明来操作远程文件。

Morris等（1986）提出Adrew分布式文件系统（AFS）。AFS最小化客户端-服务端的传输通过将整个文件从服务端传到客户端并且将文件缓存在客户端。当一个用户尝试第一次打开文件时候，会下载并且保存在客户端机器上。随后，客户端操作文件类似操作本地文件。当用户尝试关闭文件，如果文件被修改，那么修改的内容会被发送到服务器，Satyanarayanan（1989）提高了AFS的安全因素。

Hadoop（Apache 2015）是一个开源软件用于开发容错的分布式应用。Hadoop支持使用MapReduce来并行的处理大数据集。Hadoop提供了一个具有容错性的分布式文件系统即HDFS。当前市场中选择HDFS的产品有Amazon的S3（Amazon 2015），Windows Azure Blob Storage (Microsoft, 2015), GPFS (IBM, 2015), and MapRFileSystem (MapR, 2015)。

Polato等人（2014）年提出了一份调查描述了Hadoop框架在过去阶段是如何发展。报告着重指出研究社区在性能、调度、数据流、资源分配管理、生态、集群资源管理、数据安全以及加密方面的贡献。

Siddiqa等人（2016）进行了一项关于大数据管理的调查，并提出了对于流进程管理的分类。这个分类包含了最新的技术、工具以及用于数据存储、数据预处理、数据处理、以及安全的技术。可扩展性、可用性、诚实、异质性。

Dong等人提出S—HDFS一种可以存储小文件并且可以有效访问HDFS上小文件的方法。他们建议文件合并以及对于节构化相关的小文件进行schema预取；对于逻辑相关的小文件进行分组以及schema预取管理。他们的实验结果表明他们的算法模型比原生的HDFS以及Hadoop文件归档措施性能要好。

Chang等人（2008）提出一种可扩展的分布式存储系统来管理结构化的数据，称为Bigtable，是一个多维的排序图来通过行键、列键以及时间戳来进行索引。Google的许多项目（比如Google Earth，Google Finance等）都是将数据存放在Bigtable上。列族将一组或多组列键进行归类并用于访问控制。其中文件内容的版本通过时间戳来进行控制。

Apache HBase（Apache 2015）是一个NoSQL数据库是以Google的Bigtable为原型并且通过Java来实现。HBase对于存放在其上的数据模型提供了随机访问以及随机写的功能。HBase的数据模型通过一个集群来进行数据的切分以及分发，其数据模型包括表、行、列、列族、以及行键。一张表的行被切分并且存放在不同的Regions。行健是用来进行索引并且每一个行健是唯一的。每一个列族有一列或者多列，并且被存放在低一级的存储文件中，该文件叫做HFile。

用户需要将数据转化为HBase的数据模型才能使用HBase。在HBase中很难存放大的二进制文件，并且HBase只支持ACID语义在每一个行级别上。

Weil等人（2006）提出Ceph，一个可扩展并且高性能的分布式文件系统。Ceph设计将数据从元数据管理中分离出来并且允许二者单独scale。他们使用一个数据分布式功能（CRUSH）来分配对象到存储设备上，这消除了需要保持和分发对象列表的要求。他们提出的这个模型通过CRUSH功能来定位一个对象而不是使用元数据。CRUSH增强了数据的副本分离在失败区域来提高数据的安全性。

Li等人（2014）提出Tachyon，一个分布式文件系统允许可靠的数据共享以一个内存的速度对于集群计算框架。Tachyon利用了线性的概念，通过重新执行操作来恢复丢失的输出。。Tachyon使用边界算法限定数据的重新计算代价通过不断的在后台异步检查点文件。Tachyon使用优先级以及权重的占共享基于资源分配模式来重新执行任务。

MapR-FS（MapR，2015）是一个分布式、可扩展具有容错性的文件系统，使用一个container location database以及大量的containers来管理文件系统。这个container location database存储着container的位置。Container是一个副本数据结构包含目录或者文件并且具有并行性。一个典型的container大小是10-30GB。MapR-FS环境自动调节container的大小当集群空间使用高度不平衡时。MapR-FS使用一个固定的块大小（8KB），这个大小与常规文件的操作很适合。一个文件会分裂为大小为256M（可以调节）大小的chunks，这些chunks是有副本备份的。这些chunk是高可用的并且支持并行操作。Chunk的大小可以在directory级别来设置。MapR-FS可扩展是因为其可以包含的container数目以及每一个container的大小可设置。由于文件系统的元数据是通过许多的container来传输，所以对于元数据的修改可以是并行的。这也提供了高的吞吐量，MapR—FS没有单节点错误。

Cassandra（Lakshman and Malik, 2010)是一个可扩展的分布式存储系统来管理结构化数据。其提供了高可用的没有单节点错误的服务。Cassandra的分布式的多维map，可以通过key来进行索引，其值是一个高度结构化的对象。它使用hash一致性来切分数据通过集群的方式。他提供ACID语义在一个每行的基础上，并且支持Rack Unaware, Rack Aware, and Datacenter Aware副本机制。

Zhang等人（2012）提出搭建在HDFS上的分布式缓存系统叫做HDCache来支持云环境。对于每个缓存的文件都有三个副本并且保存在不同的缓存服务节点上来提高鲁棒性以及平衡工作量。缓存服务是一个对等的结构模型并且使用了分布式的hash table。HDCache系统使用共享内存作为基础，并且在共享内存中的内容可以被多个客户端线程共享。

Druid（Yang等人，2014）是一个分布式，面向列的实时的对低延时查询进行存储优化。支持流数据摄入，一个Druid集群有四种类型的节点，分别是Real-time nodes, Historical nodes, Broker nodes, and Coordinator nodes。所有的节点都是使用Zookeeper来协调。Real-time节点可以收到事件流并且在内存中进行索引，并且为了利用这些后面进行查询。该节点短时间保存这些事件然后将这些事件收集到Historical Nodes。Historical Nodes彼此是孤立的，并且他们只知道如何进行加载、删除并服务不可变的段。Broker Nodes使用发布在Zookeeper上的元数据进行规划Historical nodes和Real-time nodes的查询。Coordinator nodes定期决定集群的状态并请求Historical nodes加载新的数据，删除过时的数据，备份数据以及移动数据来均衡负载。

Amin等人（2015）提出Social Trove，一种通用的自我总结存储服务, 用于社会传感应用，这服务分层地将传入的数据流实时分组基于用户指定的应用程序特定距离度量 (即模型数据项之间的相似性)。它将集群的摘要模型缓存在内存中。这使得Social Trove可以有高的吞吐量以及低延时的查询来满足实时传感数据。

Duong等人（2013）提出切分技术来快速查询用户的信息通过给定用户的社交联系方式。他们将用户密集连接的社交关系分配给同一碎片。少数用户占大量的碎片访问量。这样的用户信息会备份到正确的碎片中，这些技能帮助提高了查询过程的速度。

Chandrasekar等人（2013）提出一种新的索引方案来有效的解决存放在HDFS上的小文件。一系列的小文件被结合来创建一个大的文件，然后把结合的文件放在HDFS上，通过索引预取NameNode的数据加载时间降低

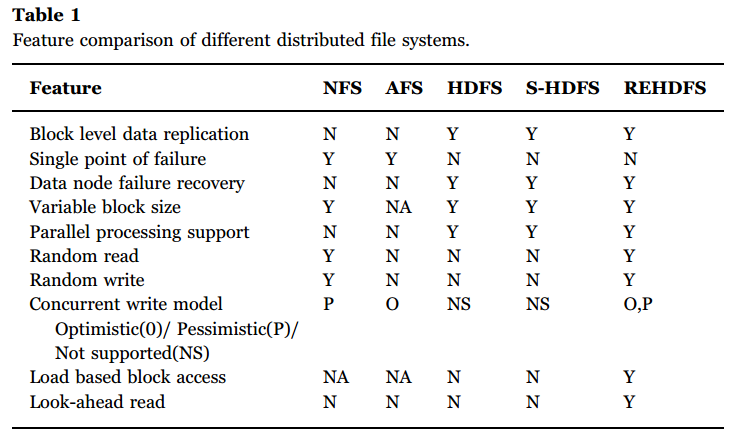
Kwon等人（2008）提出SGuard，用于容错分布式流处理进程。SGuard使用一个新的调度系统（Peace）来扩展HDFS从而降低写单独检查点的时间。SGuard使用Memory Management Middleware来实现检查点异步。

Chandakanna和Vatsavayi（2015）提出一个自我学习以及适应负载均衡器，这个均衡器定时的评估服务器的能力以及自我调整其路由基于之前内部反馈得到的性能。他们提出的方法既不需要服务器的显式定期通知, 也无需负载平衡器进行周期性轮询。

Yang和Chen（2015）提出Adaptive Task Allocation Scheduler(ATAS)来选择最好的节点来执行MapReduce任务。ATAS将节点分为quick node以及slow node。它使用节点类型来评估响应时间以及分配影响系统的后台任务。

Kung以及Robinson（1981）提出一种开放式并发控制机制。这个开放式模型适用于频繁完成任务而不影响其他任务的应用场景

NFS是一个无状态分布式文件系统协议，其不支持文件级的直接锁定。Network Lock Manager (NLM)协议（NFS的4.2部分扩展）支持文件级别的锁定。NFS 服务器端部分和 AFS 分别为NFS以及AFS文件系统创建单点故障。HDFS将这些文件块备份并且存放在不同的DataNodes，当一个DataNode挂掉了，HDFS在活跃的DataNode上创建丢失块的副本来满足约定的副本因素。HDFS是Hadoop的一部分提供了并行数据的处理模型。本文提出的REHDFS支持随机读写，提前块读加载基本数据节点选择来允许用户在当前写请求中选择乐观或悲观模型。不同的分布式文件系统NFS, AFS, HDFS,S-HDFS, and REHDFS的比较见Table 1.



提出的REHDFS分析了不同的块存放机制来支持用户的副本因素请求。它研究了快速的文件内容查询通过：

1. 智能选择DataNode来下载块
2. 同时下载块并且预取前面的块

REHDFS通过支持随机写来扩展HDFS，并且REHDFS提供了pessimistic and optimistic models（这个我不太懂是什么）来实现随机写操作。它允许用户对于HDFS的文件执行随机读写操作

# REHDFS

HDFS通过将大文件进行切分为小的块（block）并且将这些块在多个DataNodes上进行备份来实现可靠性。HDFS使用默认的块大小来切分文件，并且允许用户定义其大小。HDFS既没有规定确切的存放block策略也没有规定需要用多大的block size。REHDFS可以用来学习不同块大小带来的影响以及研究不同的块存放策略带来的优势和劣势。一旦文件被分块备份并存储在不同的DataNode上。客户端需要一种方式来透明地访问文件内容。本文提出的REHDFS评估了几种策略来实现客户端API进而实现访问HDFS上的文件内容。访问透明性是一个重要的目标对于客户端API来访问HDFS上的内容。存放在HDFS上的文件只能顺序访问而且只能追加写。**REHDFS提供了一种随机读取文件内容的方式并且提出不同的机制来执行随机写操作。**

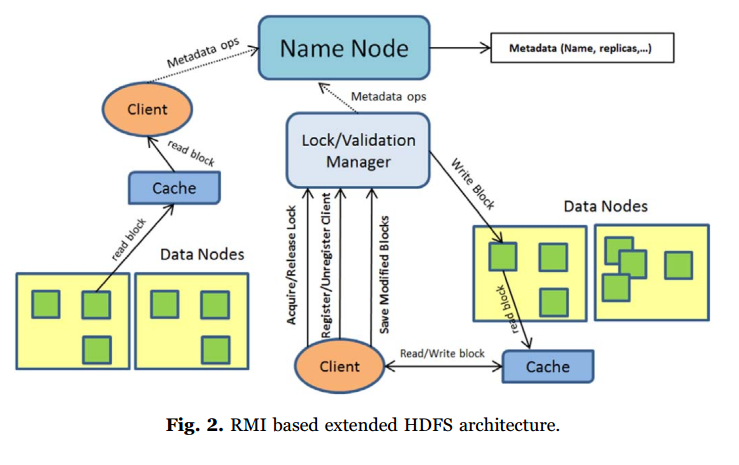
本章描述了REHDFS的架构以及存放文件在REHDFS上的细节。3.2描述了用于选择最好的节点来下载block的策略，3.3描述了随机读写的实现细节。

## 3.1 REHDFS架构及特性

这部分主要提出了REHDFS的架构以及REHDFS支持的其他特性。

REHDFS逻辑视图：REHDFS架构如Fig.2.其主要的组成部分是一个NameNode，一系列的DataNodes，一个客户端组件，一个缓存组件以及一个锁定/验证管理。

NameNode管理元数据，DataNode存储一个或多个块，最终用户通过使用Client听过的模块来访问存放在REHDFS上的文件。客户端模块隐藏了与其他REHDFS组件联系的细节并提供了访问透明度。客户端模块使用缓存模块来缓存从DataNode上检索的块。客户端发起的快请求通过缓存来提供，如果这个块正好在缓存中。否则，从DataNode来检索对应的块，然后添加到cache并且一个block副本会返回给client。当客户端修改一个文件特定的部分，与修改块相关的缓存块会相应的更新。缓存模块也追踪被客户端修改的块。



Lock/Validation Manager是添加到REHDFS的一个新的组件来支持随机写操作。模块添加了pessimistic model and the optimistic model来实现随机写操作。在pessimistic model，想要修改一个文件，客户端需要从Lock/Validation Manager获取一个锁。在optimistic model，客户端可以不请求锁来做出改变。当其他客户端正在操作一个文件的时候，其他的保存操作可能会失败。详细的介绍在3.3节。

CAP原则（https://en.wikipedia.org/wiki/CAP\_theorem 2017）由Eric Brewer提出，由Gilbert and Lynch (2002)证明。在网络分区存在的情况下，人们必须在一致性和可用性之间进行选择。在HDFS上，NameNode管理文件系统元数据（元数据就是存放比如存储位置、历史数据、资源查找、文件记录等的数据，就是记录数据的数据）。客户端需要向NameNode发送请求来获取存储文件块的DataNode位置。任何新的客户端都不能操作任何文件当网络分区隔离NameNode。HDFS客户端可以容许网络分区当NameNode以及至少一个DataNode对每一个块是存活的。即使当网络没有分区，客户端仍需要联系NameNode来获取元数据进而对文件进行操作。一旦元数据是可用的，客户端可以直接联系DataNode来检索内容。当执行写操作以后，其元数据就会被改变。REHDFS也以来NameNode来完成写操作。总结来说HDFS（REHDFS）选择consistency（CP）over availability (AP).

REHDFS设计考虑的几个方面是：位置透明，访问透明，乐观和悲观模型以及锁的粒度来支持写操作。

*Location Transparency and Access Transparency*（位置透明以及访问透明）:REHDFS提供了位置透明以及访问透明。存放在REHDFS上的文件被切分为blocks，并且每个副本有多个备份在不同的DataNodes上。NameNode是一个不同的节点在Hadoop集群上来存放每个文件的元数据。元数据包括文件系统的命名空间、块大小、文件大小以及每个块存放在DataNode的位置信息。每个存放在REHDFS上的文件都有一个特定的文件路径以一个知名的前缀（e.g. dfs://）来区分其他文件系统使用的文件路径。REHDFS文件的文件路径隐藏了给定文件的块路径细节但是给出了位置透明度。客户端模块联系NameNode来检索与给定文件相关的元数据。最终用户使用客户端模块API来操作给定的文件。客户端模块提供的API类似与本地文件系统，提供了访问透明度。

*Placing Blocks on Data Nodes:*大多数Hadoop集群部署使用了俩个或多个机架（NameNode）。每一个block需要被复制到每个机架的至少一个DataNode来获得最大的容错性从而可以允许机架的出错。本文一个客户端称为一个远程客户端（*Remote Client*）当客户端运行在一个节点时，它不是任何一个存储block的机架的一部分。否则，客户端被称为是本地客户端，而且客户端运行的机架称为*primary rack。*当把一个block的副本放在每一个non-primary 机架，剩下的block放在*primary rack不同的DataNodes*上。对于本地的客户端应用，为了避免

## 3.2 REHDFS检索

## 3.3 随机访问及随机写

# 实现

# 结果分析

## 5.1 服务器选择策略

## 5.2 与传统性能对比

## 5.3 随机访问性能

## 5.4 REHDFS读API对比

## 5.5 REHDFS写API对比

# 结论

# 参考文献