单位代码：10293　密　级：公开

南京郵電大學

硕士学位论文

论文题目：HDFS 文件系统的改进研究

学　　号	1015041230
姓　　名	周长俊
导　　师	宗平
学科专业	计算机应用技术
研究方向	分布式计算技术与应用
申请学位类别	工学硕士
论文提交日期	2018 年 6 月

The Improvement Research of HDFS File System

Thesis Submitted to Nanjing University of Posts and Telecommunications for the Degree of Master of Engineering

By

ZHOU Changjun

Supervisor：Prof. ZONG Ping

June 2018

南京邮电大学学位论文原创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京邮电大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

本人学位论文及涉及相关资料若有不实，愿意承担一切相关的法律责任。

研究生学号：_____ 研究生签名：_____ 日期：_____

南京邮电大学学位论文使用授权声明

本人承诺所呈交的学位论文不涉及任何国家秘密，本人及导师为本论文的涉密责任并列第一责任人。

本人授权南京邮电大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档；允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索；可以采用影印、缩印或扫描等复制手段保存、汇编本学位论文。本文电子文档的内容和纸质论文的内容相一致。论文的公布（包括刊登）授权南京邮电大学研究生院办理。

非国家秘密类涉密学位论文在解密后适用本授权书。

研究生签名：_____ 导师签名：_____ 日期：_____

摘要

对于云端存储的海量数据来说，提升分布式文件系统的性能以及云端存储数据的安全性与可问责能力是必要的。然而，HDFS 默认的副本存放策略进行副本存放节点的选择时具有随机性，那么随之而来的问题是节点间副本存放不均衡以及数据恢复时由于距离因素造成内部带宽的巨大消耗；与此同时，对于需要将海量数据寄存在云服务平台上的用户来说，HDFS 未能提供足够的安全机制来确保数据的安全性。因此，针对 HDFS 默认副本存放策略以及如何提供安全的数据存储及操作环境的研究是有意义的。

本文在对 HDFS 进行研究与分析的基础上，从默认副本存放策略以及数据安全性两个角度来考虑改进工作。

针对 HDFS 默认副本存放策略中存在的不足，改进后的副本存放策略将节点之间的距离、节点当前的负载情况、节点磁盘 I/O 效率以及副本失效次数等因素纳入节点选择的考虑范围内，并依此计算出每个节点的匹配度，随后选出匹配度最高的节点作为远端机架间的副本存放最佳节点。实验结果表明，不但实现了节点间副本数目的负载均衡，而且兼顾了数据恢复时消耗的内部带宽；由于将数据副本失效次数纳入考虑因素，可以实现经常失效数据副本的快速恢复。

为了应对云存储中数据安全性问题，本文设计了一种基于可信第三方的 Kerberos 审计问责方案，在确保用户数据安全性的同时，提供了发生数据安全问题时的审计问责能力。该方案通过采用用户认证流程、数据交互以及审计与问责的机制实现了云端数据的安全性以及审计问责的能力。

关键词：HDFS，副本存放策略，Kerberos，数据安全，审计问责

Abstract

It is necessary to boost the performance of distributed file system and the security level and accountable capability of data stored in cloud. However, according to the default HDFS replica placement policy, the placement node selection is random, and then the following issues are uneven replica placement and big consumption in internal bandwidth due to the distance factor when the data needs to be restored. In the meantime, for the users who saved mass data on the cloud-based platforms, HDFS cannot provide sufficient security system to ensure data security. Therefore, it is very necessary to make researches on the default Hadoop replica placement policy and how to provide secure data storage and operation environment.

Based on the research and analysis of HDFS, this thesis makes improvements from the perspectives of default replica placement policy and the security of data.

This thesis proposes the measures to improve the deficiency of the HDFS default replica placement policy, which can take a series of factors including the distance between nodes, the current node loading situation, the I/O efficiency of the node disk and replica restoration times after optimization into consideration, calculate the matching degree for each node and select the node with the highest matching degree as an optimal node to place replica between the remote racks. The performance test result of the backup data placement policy after optimization indicates that not only load balance of the backup data placement between nodes has been realized but also the internal bandwidth during the data restoration process has been taken into account. As the times of invalid replica has been taken into account, the rapid restoration of the frequently invalid replica can be realized.

This thesis designs a reliable third Kerberos accountability solution, which can ensure the data security and audit accountability for data security issues. The reliable third party Kerberos audit accountability solution can ensure security and audit accountability of data on cloud by the user authentication process, the data interaction and audit accountability mechanism.

Key words： HDFS, Replica placement policy, Kerberos, Data security, Audit accountability

目录

第一章 绪论

随着当代社会信息化程度的持续提升，数据呈现爆炸式增长趋势[1]。海量数据如何实现安全有效存储是一个亟需解决的问题。诚然，目前已有许多云存储服务商通过应用分布式文件系统搭建的云存储平台为用户提供高存储量以及高可靠性的数据存储与管理，但针对分布式文件系统架构设计中存在的问题提出改进方案来提升对外提供服务的能力以及实现云端数据存储的安全性仍是十分必要的。

1.1 研究背景和意义

云存储[2]概念的提出可追溯至云计算[3]的发展，既然需要实现海量数据的存储，那么显然单台机器无法满足数据存储的需要，因此需要多台机器构成的集群；集群中的机器之间需要通过分布式文件系统来实现数据的有效存储以及对外提供数据访问的能力。

云存储由存储设备、网络设备、应用软件以及客户端组成。这些组成部分都间接地与存储部分相联系。应用软件实现的功能是向用户提供数据的存储以及访问操作。云存储系统的基本结构模型[4]自上而下分别为应用访问层，应用接口层，存储管理层以及物理层。图 1.1 为云存储系统的基本结构模型：

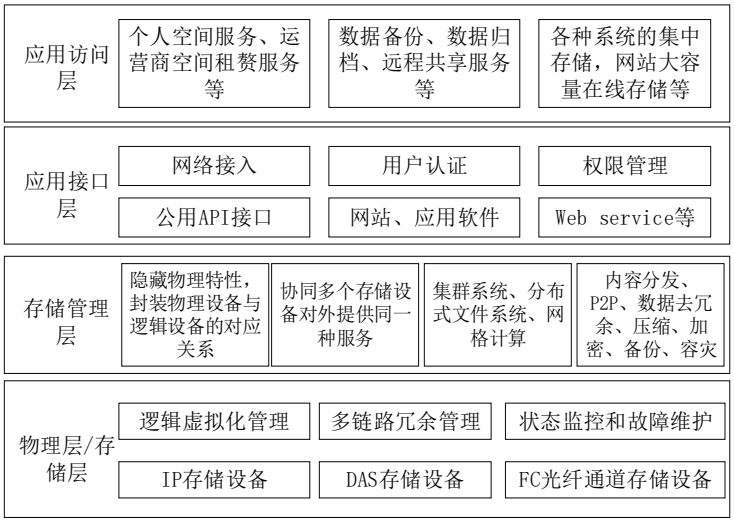

图 1.1 云存储系统的结构模型

（1）应用访问层是面向用户的。云存储系统对用户来说主要功能是进行数据存储以及数

据访问。由于云存储服务商服务类型不同，设计思路也有所不同，故而用户访问云存储系统的方式以及访问类型也存在区别。

（2）应用接口层通常是由云存储服务商来负责开发实现的，通常情况下需要按照用户需求的业务类型，开发出对应的应用服务接口，实现用户应用服务的定制化，满足用户的需要。

（3）存储管理层是最核心的部分，通过分布式文件系统来实现集群中所有节点的协同作用，一致集群中的所有节点对外提供服务方式的同时能够提升系统对外的服务能力。

（4）物理层是最为基础的部分，由于云存储中通常包含多个存储节点而且这些节点的分布较为分散，节点之间需要通过网络连接。为了实现上述节点的统一管理、监控以及故障的发现与维护需要在物理层之上增加软件系统。

1.1.1 副本技术

由于云存储系统需要存放海量数据，因而相比较于传统文件系统来说，云存储系统通常情况下需要由数百甚至更多的节点组成。为了实现商用，几乎这些节点都是廉价的，因而出现病毒侵入、网络中断、硬件设备故障以及无法预知的自然灾害都是比较常见的，一旦发生上述状况，数据的损坏或丢失在所难免。为了保证数据的高可用以及高可靠，现有的通常做法是将文件的多个副本分布于集群之中的不同节点上，这就是副本技术[5][6]。副本技术对云存储系统的影响主要体现在以下三点：

（1）可提高系统的可靠性；

（2）能够平衡并发访问时对数据节点的负载；

（3）可提高系统的访问速率。

除副本技术外，业界还有一些其他的相关处理策略，如无副本策略、最佳客户策略、瀑布式策略和普通缓存策略等。

无副本策略通常情况下也称之为缓存策略，也就是说不需要为文件通过复制而形成副本，而是采用将所有的文件存储到系统环境下的某个机器当中的一种处理策略。

最佳客户策略实现的依据是通过对文件访问历史记录的分析做出相应的处理，而文件访

问的历史记录是通过系统环境下每一个节点所记录的该节点所包含文件的被访问次数以及访问该文件的节点在系统环境下的编号。在具体实现方法中，首先需要设定一个阀值，用于判断文件的热度，一旦系统中某个节点中的文件被访问次数超过规定的阀值，那么就需要在该节点上增加该文件的份数，通常情况下是增加一个副本，之后需要清空上文提到的文件历史访问记录。这种操作的发生一般是通过定时任务来实现，通过阶段性的分析来判定文件的最佳客户。循环往复上述过程，周期性的判断系统中是否存在某个文件的最佳客户。由于通过增加热点文件一个副本，所能够带来的好处就是能够提升文件的读/写效率。但是，由于这种副本创建行为是云端自身发起的，所以可能并不能真正有效地反映用户需求，因而也可能会造成磁盘资源利用不合理的现象。

瀑布式策略应用的对象是需要实现分层结构的存储，应用三级瀑布的思路，通过统计系统中根节点被访问的次数，一旦发现超过某一特定值，那么需要从根节点的下一级的所有节点中选择最合适的节点作为文件副本存放的节点，若该节点被访问的次数也已超过某一特定值，那么需要选择客户端作为文件副本存放的节点。通过分层来实现分担文件读/写的压力，在提升系统服务能力的同时降低了单个节点的服务压力，但其应用受到一定的限制，只能应用于分层结构的存储系统，因此具有一定的局限性。

普通缓存策略是当系统中某个节点中的某个文件需要被读/写时，该节点需要通过复制该文件产生一个副本并将其存放于所在节点之上。这种方式通过增加文件的份数来实现读/写操作压力的分担，但是如果读写的是大文件，那么复制产生副本的过程对节点磁盘 I/O 的消耗也可能是巨大的，同时每次访问就增加副本对节点的磁盘容量将会是一个巨大的考验。

1.1.2 数据安全技术

历史数据在企业发展中所起的作用越来越大，企业对于海量数据存储的需求日益增加，由于受到资金以及技术等方面的制约，越来越多的中小型公司逐渐选择将其私密数据存储至云端。对于这些公司来说，将私密数据存储至云端，一方面实现了数据的有效存储，为后续的使用提供便利；另一方面，降低了自身建设云存储环境的成本，能通过相对较小的代价实

现海量数据的存储。诚然，云存储服务商的出现给有海量数据存储需求的公司带来了福音，但是一旦用户选择将数据存储到云存储服务商提供的服务器之上，用户原本具有的数据隐私性保护也会随之而丢失。由于无法判断云存储服务商是否可能泄露用户数据或者当云存储服务商由于自身系统发生问题而导致数据丢失或外泄的现象，所以云端数据的安全性如何保证将会是一个亟需解决的问题[7]。

事实上，早在 2009 年 Gartner 公司做过一项调查，结果显示：受访企业中约占 70%的 CTO 拒绝采用云存储的缘由是担心数据的隐私性以及安全性；同年 3 月 7 日，互联网巨头 Google 公司就因为文档的缘故而导致了用户文件泄露的事情。因此，保证云存储数据的安全性是迫在眉睫的事情。

云存储中数据安全性涉及两个方面，一个是云存储服务商，另一个是云存储用户，所以云存储中数据安全性的特点[8]主要呈现在两个方面：

（1）云存储服务商是否完全可信无法判断，可能存在为了自身利益而做出危害用户的数据安全性的行为。当然，云存储服务商为用户提供存储数据的安全性是其义务，但是由于缺乏合理有效的监督机制，无法确保云存储服务商是否按照协议的要求有效的保证存储数据的安全性。而且云存储服务商自身的自然灾害和机械故障而导致数据的泄露甚至丢失都是有可能的，在这种情况下，如何判定双方责任也缺乏有效的机制，可能导致责任推卸现象的发生。

（2）云存储过程中所产生的一系列动作如何记录作为事后判定责任的依据。虽然用户存储数据会与云存储服务商事先签订相关协议，但一旦用户发现存储至云端的数据疑似出现了丢失、泄露以及损毁的情况，由于缺乏有效的依据而无法实现对事故双方的责任进行判断，需要存在一种机制，能够实现一方面维护用户数据隐私的权益，另一方面减少云存储服务商被别有目的的用户诬陷的可能。

访问控制技术是保证数据安全性的一种重要技术[9]。访问控制技术的主要功能是实现禁止没有授权的用户对相应数据的访问。访问控制技术发展至今主要分为以下三种机制[10]：

（1）自主访问控制。自主访问指的是一旦自身拥有访问权限，那么随之也就拥有授予或者取消其他用户自身所拥有的访问权限的能力。自主访问控制中常用的安全机制包括访问控

制矩阵以及访问控制列表。

（2）强制访问控制。强制访问控制指的是用户只能按照系统事先设定的针对每个用户的特定等级来进行相关资源的访问，用户自身无法实现访问权限的修改，系统管理员进行相应的设定是增加或删除用户权限的唯一途径。

（3）基于角色的访问控制。基于角色的访问控制中角色一词是理解用户以及权限的基础，用户以及权限都是通过角色发生联系的；权限又可以具体理解为用户拥有什么样的角色，而该角色又赋予该用户对目标进行何种操作。基于角色的访问控制规定了用户、角色、操作权限和目标文件之间的约定关系。这种方式不仅使得权限管理愈加灵活以及可扩展，而且可降低了权限管理的复杂度。

消息加密[11]指的是通过某种方式将明文消息转变为密文消息。明文转换为密文的方式是对消息进行加密，反之则为对消息进行解密。基于密钥的加密算法主要分为以下两类：

（1）对称加密算法。在该算法中，加密和解密密钥可以互推，发送者和接受者通常会事先商量一个密钥。对于该算法来说，只要能够获取密钥，就能实现密文的解密，因此密钥的保管对其来说至关重要。

（2）非对称加密算法。该算法分别包含加密以及解密这两个密钥。与对称加密算法不同的是，密钥之间无法实现互推。加密密钥是可以被公开的，其是否公开对于消息的隐秘性不会产生任何的影响；解密密钥是不可以公开，一旦解密密钥被窃取或者被泄露那么消息就容易被破解，因而也就很难保证数据的隐秘性。

副本技术的发展一方面能够应对云端数据突发灾难时可能存在的数据丢失现象，另一方面提升数据副本数目能够加快用户请求响应速度与提升用户体验；数据安全技术的发展则有利于实现云端数据存放的安全性，通过加密算法加密存储于云端的用户数据可以降低用户数据泄露的风险，记录用户对云端数据的操作用于数据安全问题发生时的责任判定。因此，副本与数据安全技术的发展对于云存储的发展来说起着至关重要的作用。

1.2 国内外研究现状

目前有关云存储的研究工作比较多，也有不少成功的案例，下文着重讨论主要的云存储案例。

1.2.1 云存储系统

1. Google File System

Google File System（GFS）[12]是 Google 公司推出的常用分布式文件系统之中的一种，具有高可扩展性，适用于需要访问海量数据的应用。GFS 系统中所采用的节点均为廉价的商用计算机，可以根据系统的需要向其中添加任意数目的廉价商用计算机用于满足数据存储的需要，实现了减少运营成本的同时能够对外提供优良的服务。GFS 系统架如图 1.2 所示：

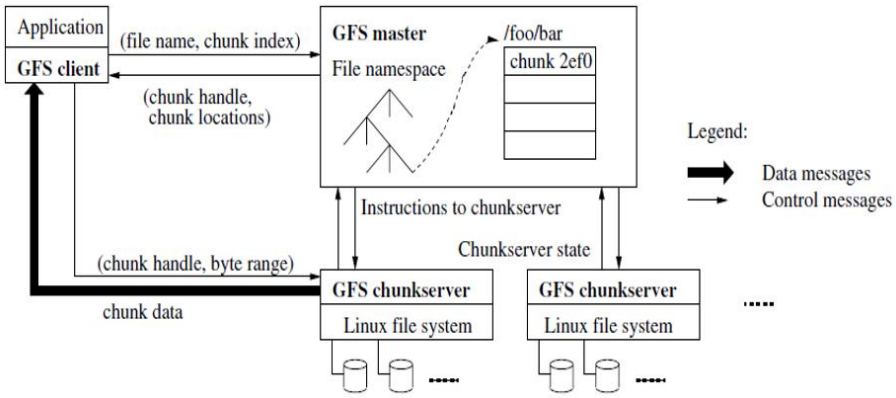

图 1.2 GFS 系统架构

GFS 中的所有节点只能被赋予如下三种角色中的一种：

（1）总控服务器：主要职责是维护 GFS 系统的元数据信息以及控制整个系统。元数据信息包含块空间名称、文件到块映射表，块所在节点的位置信息等；其控制能力主要体现在块的租约管理、块的复制以及回收无用块等。为了获取系统中块节点的可用情况，总控服务器通过心跳检测机制来判断块服务器是否处于正常工作的状态。

（2）数据块服务器：对应于系统中用作数据存储的节点。对于 GFS 来说，将文件分割成大小为 64MB 的块进行存储。为了避免出现单个数据块服务器负载过高的情形因此在存放数据块时需要考虑节点当前磁盘剩余空间以及最近新建数据块的数目等因素。为了避免删除

所带来的节点磁盘 I/O 的占用，采用的是后续新数据块覆盖的策略来实现文件块删除操作。

（3）客户端：是用户实现与 GFS 交互的接口，当用户需要访问 GFS 中的某个文件时首先需要做的不是直接访问数据块服务器而是访问主控服务器，获取存储该文件的所有数据块服务器中离客户端最近的节点，然后客户端访问该数据块服务器来获取所需要读取的文件。

GFS 系统对于文件的存储与传统的文件系统不同[13]。对于传统的文件系统来说更多情况下存储的是大量的小文件，可是对于 GFS 来说则是更多的是存储少量的大文件；它将文件分割成为若干个大小固定（64MB）的数据块，为了实现唯一标识这些数据块方便后续的查找工作，总控服务器会选择为这些数据块创建一个 64 位的唯一标识。同一文件的所有数据块会分布于系统中的不同节点之上，为了保证数据的可靠以及可用，通过复制实现系统中包含同一数据块的多个副本，通常情况下，复制因子为 3。

2. Andrew File System

Andrew File System（AFS）[14]是美国 Carnegie Mellon University 大学开发的一种分布式文件系统，同样为常用的分布式文件系统之中的一种，其设计目标是为了提供可靠以及可用的文件读/写环境。与传统文件系统所不同的是，AFS 应用于分布式环境之中，它主要的作用就是将分布于不同地域上的不同节点组织成一个完整的数据存储系统。与常见的客户端/服务器模型不同的是，系统中对外提供服务能力的节点并非一台，而是系统中的每一个节点都在对外提供服务。更为重要的是，相比于传统文件系统所依赖的系统环境，AFS 能够实现跨平台访问的能力。

在 AFS 系统中，能够实现用户共享其中存放的文件，用户无需了解文件最终存放于系统众多节点中的哪一个，也无需掌握文件存储节点所采用的是何种操作系统，这一切对用户来说都是透明的，用户只需通过 ASF 给出的逻辑地址即可实现文件的获取或者对文件进行相应的操作[15]。

AFS 相比较于其它分布式文件系统具有安全性能非常高的特性，其安全性能是通过访问控制列表和 Kerberos 鉴权[16]等手段来实现的，保障用户数据的安全性。用户试图登录 AFS 系统中进行相关操作的前提是需要通过身份验证，只有验证通过，方能判定为合法用户，才

可以访问对应网孔，否则拒绝提供服务[17]；用户是否能够对某个文件进行读/写操作需要取决于访问控制列表中是否指定用户拥有对该文件的读/写权限。

AFS 的通常应用场景为大型分布式环境，其实现目标是高扩展性以及安全数据访问，相关的关键技术包括以下三点[18]：

（1）整个文件缓存策略。当操作一个文件时，直接从服务器端读取整个文件，缓存到本地磁盘中，后续操作直接在本地执行，达到减少服务器工作量的目的。

（2）通过采用回调机制来保证客户端所缓存的数据与服务端存放数据的一致性。所谓的回调机制是服务器端主动告知客户端某些文件是否已经过时，而并不是客户端以轮询的方式询问服务器端。这种方式不但实现客户端与服务器端之间交互次数的降低，而且也减轻了服务器端的负载压力。同时，由于整个文件全部缓存，即便出现网络状况不佳甚至服务器中途重启的状况，那么也不会影响提供 AFS 文件服务，系统可用性自然就提高了。

（3）AFS 提供的是基于网孔结构的分布式文件系统，其明显的优势在于便于管理。若要实现用户与网孔外单元格内的用户之间的共享信息，那么需要获得其它网孔管理机构授予用户的网孔访问权。对于传统文件系统来说，提供用户文件目录进行访问；对于 AFS 来说，则是通过提供逻辑路径来实现，此路径具有唯一性并且对用户来说是完全透明的。通过逻辑路径，用户与访问文件目录一样。

3. Taobao File System

Taobao File System（TFS）[19]是中国阿里巴巴公司推出的一款国内分布式文件系统的翘楚，主要实现的是海量小文件的存储。一般情况下文件大小不大于 1MB；部署于 Linux 系统之上，运行于淘宝网之上，高效、可靠地实现淘宝网的海量数据的存储工作。TFS 系统架构如图 1.3 所示：

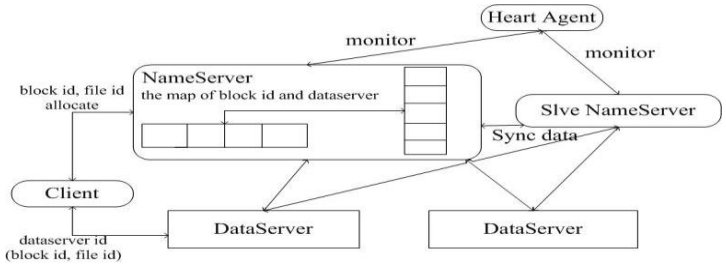

图 1.3 TFS 整体架构

TFS 集群通常情况下由两个 NameServer 以及任意多个 DataServer 组成，之所以选择两个 NameServer 主要是为了容灾机制，一旦其中一台 NameServer 宕机后立即切换至另外一台对外提供服务[20]。

TFS 中存储的小文件并不是零散存放的，而是通过将众多的小文件聚合成一个大文件，通过块将将其组织起来。TFS 中将块分为主块、扩展块两类，其中主块负责聚合的大文件的存储，扩展块用于剩余的小文件的存储，这种设计方式能够更加有效的利用磁盘空间，主块的大小通常远大于扩展块[21]。每一个块都会在 NameServer 对 DataServer 划分磁盘空间时分配唯一标识的编号，其映射关系有 NameServer 负责进行维护，也就是说查找时需要依赖 NameServer 而真正数据存储则依赖于 DataServer。TFS 集群中的每一台 DataServer 都会开启多个独立的进程，每一个进程负责该 DataServer 中一个挂载点的写入，这种方式不但提升了写入的性能，而且一旦某个磁盘发生损坏也不会进而影响该台 DataServer 的服务能力。

NameServer 除了负责维护块到 DataServer 之间的映射之外，还包括：TFS 集群中 DataServer 的增加与减少、通过心跳机制检测 DataServer 服务状态以及实现集群的负载均衡。

DataServer 存在的目的是为了实现存储数据以及数据读写。

分布式文件系统中不得不考虑的问题就是容灾能力，由于机器众多，怎么实现突然宕机情况下还能正常对外服务是其必须要解决的。TFS 系统中，对于 NameServer 来说，通过采用一主一备，互为热备，同时运行，其中主机绑定对外提供服务的 IP 地址，一旦主机突然宕机则迅速将对外提供服务的 IP 地址绑定至备机，将其切换为主机，负责对外提供服务[22]；对于 DataServer 来说，通过副本来解决 DataServer 宕机后导致的数据丢失问题。

1.2.2 副本管理技术

副本管理技术按照是否实时调整副本的数目以及位置可以分为静态副本管理策略和动态副本管理策略[23]。

静态副本管理策略是指系统运行过程中，除数据删除操作之外，数据的副本数目以及存放位置不会发生变动。依据上述特性，可以发现该策略对于副本的管理以及维护的实现相对

简单，随之而来的问题就是对于系统中数据的实时状态不能做出及时响应。

动态副本管理策略是指在系统运行过程中，依据数据访问数量以及网络状况的实时变换，副本的数目以及存放位置动态变化。分别由以下两部分组成：

（1）副本创建：动态增加热点副本的数目。

（2）副本删除：动态减少冷门副本的数目。

针对 HDFS 系统中不同数据类型需要采用不同的副本管理策略，近些年提出的主要动态副本管理策略如下：

Zhendong Cheng 等人提出一种弹性副本管理系统（ERMS）[24]。基于提出的活跃/备用模型，通过一个复杂的事件处理引擎来区分实时的数据类型，对热点数据增加副本数目，而对冷门数据降低副本数目。

K. Sashi 等人提出基于数据网格环境下副本创建和放置的算法[25]。按照文件的热度来动态创建副本从而提升数据的可用性，与此同时减少热度低的副本数据，降低磁盘空间的浪费。将存放副本的节点纳入考虑范围之内，选择适合的节点进行副本存放，从而实现后续对该副本读写性能的提升。

Krish K.R.等人提出 Hadoop 的动态管理系统（AptStore）[26]。对于用户频繁访问的数据采用标准 DAS 存储，提升 I/O 吞吐率；对于用户偶尔访问的数据采用 NAS 进行存储，保证容错性。通过提出的热度预测算法来分析云存储系统日志，依此来预测不同文件的最佳存储方式。

通过分析可以发现，上述改进措施都是通过实现动态副本管理策略来改进 HDFS 默认静态副本管理策略。虽然通过副本热度动态增减副本数目以及根据用户访问数据的频率采用不同存储方式能够提升副本读写性能，但是，动态增减副本数目需要考虑集群内部存储空间的有限性以及副本频繁创建删除所带来的集群内部带宽的消耗，因此，上述改进策略依旧存在进一步改善的空间，这也是本文研究工作的价值所在。对于 HDFS 默认副本存放策略的改进，与其亡羊补牢，不如未雨绸缪，可以对副本存放节点的选择上做出优化，本文考虑通过改变 HDFS 默认副本存放策略中远端机架副本存放节点的选择策略，由于其选择远端机架节点具

有随机性，可能造成节点间负载不均衡、数据恢复以及读写时内部带宽的消耗问题。尝试通过采用本文提出的节点匹配度评价方法获取目标节点的匹配度，选择匹配度最高的节点作为远端机架副本存放节点。

1.2.3 云端数据安全性及问责机制

显而易见的是，在云存储环境下，需要通过将云数据服务与数据安全问责技术结合来实现完善的数据服务机制。

Andreas Haeberlen[27]和 Siani Pearson[28] 提出云端问责机制实现的基本要求、整体的设计框架以及后续发展中可能存在的挑战。Ryan 等人[29]完善 Andreas Haeberlen 和 Siani Pearson 提出的设计框架的同时提出云端问责机制的生命周期以及三个抽象层。Volkmar 和 Anderson Santana 等人[30]提出通过丰富安全机制以及严格业务流程的云端问责机制的实现。Jun Zou，Yan Wang 和 Mehmet A.Orgun 等人[31]通过深入研究动态逻辑系统，将其与云端问责机制相结合，实现基于动态逻辑的云端问责混合系统。

云端数据问责机制的实现需要满足下列三个条件：

（1） 能够及时准确的发现云端数据存在的问题；

（2） 能够准确无误的对出现的问题进行责任判定；

（3） 责任判定依据的凭证具有不可抵赖性，云端以及用户需要服从责任判定。

由此可知，通过提出了云端问责机制的生命周期、抽象层以及通过动态逻辑系统的研究来实现云端问责混合系统，可以在一定程度上明确了云端问责机制的实现目标，但是，对于云端问责机制具体实现方法的论述却偏少，因此本文对于云端问责机制具体实现方法的研究是有意义的。本文通过对现有云端数据安全性技术、Kerberos 认证流程的执行方式以及上述理论进行的深入研究，设计出一种基于可信第三方的 Kerberos 审计问责方案。本文方案以 Kerberos 系统中的 KDC 作为可信第三方，修改 KDC 端的原先设计，改变原先数据传输方式以及简化认证流程的方式来实现审计问责功能，该问责方案能够切实可行的保障云端数据的安全性以及发生数据安全问题时责任判定。

1.3 本文研究工作与组织结构

本文的主要研究工作有：

（1）在研究云存储现状的基础上，着重关注副本技术以及数据安全性技术发展，阐述了国内外当前的研究现状，指出副本技术在云存储中的必要性以及 HDFS 默认副本存放策略存在的主要问题。

（2）提出对 HDFS 的默认副本技术的改进思路：采用节点匹配度评价方法，对副本存放所在远端机架的目标节点计算其匹配度，按照匹配度由高到低对目标节点列表进行排序，选择最大值对应节点作为副本存放的远端机架节点，有效的解决了默认副本存放策略采用随机选择远端机架节点存放导致的节点间副本数目负载不均衡以及内部带宽的消耗；通过阶段性分析副本失效次数据此调整副本存放位置来实现经常失效副本的快速恢复。

（3）考虑到数据安全性在云存储中的不可或缺性，针对目前已有的安全问责基本模型，提出了一种改进的问责模型，该改进方案在 Kerberos 认证流程的基础上，引入可信第三方的概念，设计了基于可信第三方的 Kerberos 审计问责方案。该改进方案依赖于 Kerberos 基础架构，以 KDC 作为可信第三方，并对 KDC 的功能性做出了修改，通过存放于 KDC 端以及云端的 Ticket 记录用户对云端操作，与此同时，Ticket 也是数据安全问题发生时审计问责的依据，附于其上的三方签名保证了不可抵赖性，可以准确无误的进行责任判定。

（4）实验对比改进的副本存放策略与默认副本存放策略，通过实验数据分析证明改进的副本存放策略思路的正确性以及有效性。

本文的组织结构如下：

第一章绪论：论述了云存储系统的研究背景以及研究意义，国内外相关技术的研究现状，阐述了副本技术、安全问责模型以及默认副本存放策略存在的问题，说明了本文的主要研究工作和论文的组织结构。

第二章副本管理相关技术：论述了 HDFS 的基本概览、实现副本管理所依赖的技术以及默认副本存放策略，进而分析所存在的问题，最后通过借鉴 GFS 以及 TFS 所采用的副本管理技术的基本思想，为后续的改进工作提供思路。

第三章 HDFS 副本存放策略的改进：在对 HDFS 默认副本存放策略分析研究的基础上，针对其中存在的不足之处，提出一种改进副本存放策略，阐述了改进的副本存放策略中进行节点选择时所需的节点匹配度计算方法并具体说明了改进后副本存放策略算法的处理流程与主要的源码实现。

第四章基于可信第三方的 Kerberos 审计问责方案：在讨论云端数据安全性技术的基础上，着重对 Kerberos 进行研究，依据其认证流程的执行方式，设计了一种基于可信第三方的 Kerberos 审计问责方案。该方案选择 Kerberos 基础架构，引入可信第三方的概念，通过修改 KDC 端的设计、传输数据的加密方式以及适度简化认证流程的方式来实现审计问责功能，并从用户认证流程、数据交互、审计问责以及安全性分析等方面对方案进行了具体说明。

第五章实验结果与分析：给出了实验所构建的软件与硬件环境，搭建的源码编译环境以及 HDFS 集群环境。通过实验对比分析本文提出的改进副本存放策略与默认副本存放策略，通过分析数据块在机架以及节点之上的分布情况，据此判断改进副本存放策略是否实现预期的目标，通过对比改进后副本存放策略与默认副本存放策略存放相同数目数据块所耗时间，据此判断改进副本存放策略对文件上传时间的影响。

第六章总结与展望：总结了本文所作的研究工作，同时进一步展望后续研究的主要方向。

第二章 副本管理相关技术

分布式文件系统 HDFS、GFS 等通过其架构设计以及心跳机制等实现副本管理策略，本章主要通过对 HDFS 基础架构、数据流、容错机制以及副本管理技术的研究与分析，讨论其中存在的不足，为后续副本管理策略的改进奠定技术基础。

2.1 HDFS

Hadoop 分布式文件系统(HDFS)被设计成适合运行在通用硬件（commodity hardware）上的分布式文件系统[33]。HDFS 非常适合具有高吞吐量的数据访问需求以及大规模数据集的应用，通过对 POSIX 约束的部分放宽来实现了数据流式访问的目的。

2.1.1 HDFS 的设计

HDFS 的起源可以追溯至 Apache Nutch 搜索引擎项目，当时是作为其基础架构而开发的，目前是 Apache Hadoop Core 的组成部分。对于 HDFS 来说，通用硬件和数据的流式访问[34-35]的两个特点需要着重关注。

（1）通用硬件：相比于那些需要运行于昂贵、可靠硬件之上分布式文件系统来说，HDFS 在设计之初就被设计运行于普通硬件集群之上，这种实现方式对于集群部署来说是成本上的巨大节省也利于 HDFS 的推广应用。当然，采用普通硬件就不得不考虑其带来的节点发生故障的概率相较于昂贵、可靠的硬件显然高很多的问题，可是 HDFS 通过设计可以让用户察觉不到节点故障，保障了服务的可靠性。

（2）数据的流式访问：高效的访问模式是一旦写入，之后不再变更；后续只是对数据集进行读取操作。通常情况下，HDFS 之上的应用更多的是对存储于其中的数据进行分析（分析的是数据集中的全部数据或者是大部分数据），因而与传统文件系统相区别的是，对 HDFS 来说读取整个数据集的时间延迟比读取数据集第一条记录的时间延迟重要的多。

HDFS 系统架构[36]如图 2.1 所示：

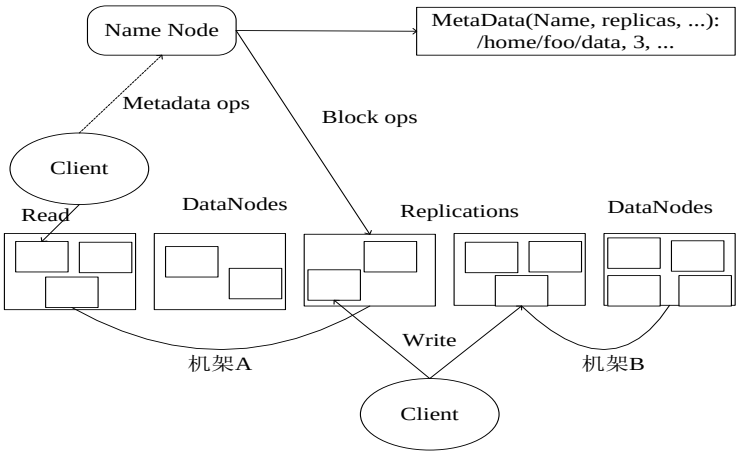

图 2.1 HDFS 整体架构

当然，HDFS 并非被设计成了满足于各种情形的分布式文件系统，依然存在许多不适用于 HDFS 之上的应用领域，相关不足之处主要体现在以下三个方面：

（1）对数据访问要求低时间延迟：由于 HDFS 设计之初的目的就是高吞吐量的数据访问需要而不是为了提供低时间延迟的数据访问的分布式文件系统，因此有此需求的应用不适宜选用 HDFS。对于数据访问有低时间延迟需求的应用可以尝试采用 Hbase。

（2）海量小文件存储：HDFS 设计之初的目的就是大规模数据集的存储需要而不是为了海量小文件存储的分布式文件系统。由于 Namenode 负责维护文件到 Datanode 的映射关系且将其存放于内存之中，若 HDFS 中存放海量的小文件，则意味着 Namenode 中需要存放海量的映射记录，按照每一条映射记录约占用 150Byte 的存储空间计算，一旦映射记录达到亿级甚至十亿级则会超出 Namenode 的内存限制，因此对于 HDFS 来说不适用于海量小文件的存储。对于有着海量小文件存储需求的应用来说可以选择 GFS。

（3）不同用户写入、任意位置修改：对于 HDFS 来说，存储于其中的文件不支持不止一个用户写入的操作，也就是说一个文件只能拥有一个用户写入。与传统文件系统相区别的是，HDFS 只支持对文件的追加操作而不支持任意位置的写入操作。

2.1.2 数据块

对于磁盘来说，其进行读写操作的最小单位为数据块。数据块是操作系统为了操作磁盘而虚构出来的概念，默认情况下数据块大小为 4k。

对于 HDFS 来说，由于其存放的通常为大规模数据集，可能存在单个节点无法满足存储需要的情况同时也是为了后续并发处理的需要，因此其采用分治的思路，将大文件分割成多个小文件，这些小文件被称为块，默认情况下，每一个块的大小为 64MB。对于那些不能被 64MB 整除的大文件，剩余的小于默认块大小的文件不会占用整个块[37]。

通过对比可以发现，HDFS 中块的大小远远超过磁盘数据块的大小。对于一次文件操作来说，首先需要做的就是组成文件块的定位，其次才是块的传输操作，一旦块比较大，那么相对于传输时间来说定位时间就明显要小的多，因此本次文件操作的时间决定于磁盘的传输速率。

分布式文件系统 GFS、TFS 中也都包含块的概念，之所以对文件进行抽象有如下原因：

（1）由于硬件的限制，可能存在某个超过单个磁盘文件大小的文件的存储需要，通过块的概念则可以对大文件进行分割，存储于集群的多个节点之上，满足存储的需要。

（2）通过对块概念的抽象，可以将文件原本所具备的存储以及元数据信息相抽离，这种方式可以简化每一个模块所需要实现的功能，Datanode 只需要负责块的存储而不需要考虑块所属文件的元数据信息，元数据信息则由 Namenode 负责管理。

（3）分布式文件系统中单个节点失效是经常发生的情况，如果没有块概念的抽象，单个节点失效则会导致整个文件的丢失，固然副本的存在不会导致文件的永久丢失，但整个文件的恢复耗时相较于文件中的某个块恢复来说是及其耗时的。同时，块概念的抽象对读性能所带来的提升是显著地。

2.1.3 Namenode 和 Datanode

HDFS 集群中的所有节点按照角色划分可以分为两类：Namenode 和 Datanode。通常情况下，HDFS 集群包含两个 Namenode（一主一备）和任意数目的 Datanode。Namenode 与 Datanode 的运行方式为 Master-Slave [38]。

Namenode 负责维护整个文件系统的名字空间(namespace)，将信息以文件形式存放于节点的磁盘之上。除此之外，Namenode 还负责保存文件的各个块到所在 Datanode 之间的映射信

息，为了提升响应速度，将信息存放于内存之中，由于内存大小的限制，因此整个文件系统所能存放的文件数目也是有限制的，不适合海量小文件的存储工作。

Datanode 是文件各个块的最终存储节点，其主要实现数据块的读/写操作，为了实现 Namenode 准确存放数据块到所在 Datanode 之间的映射，通过心跳机制，Datanode 定期向 Namenode 汇报当前存储的数据块的信息。

Client 即用户接口，用户通过 Client 可以实现文件的读/写操作。

显然，Namenode 在 HDFS 中占据着至关重要的作用，一旦 Namenode 节点失效，则意味系统的崩溃，对于如何解决上述问题，HDFS 设计者提供了如下两种思路：

（1）通过将恢复文件系统所需的元数据持久状态信息实时同步至一个远程挂载的网络文件系统当中。这种方式的优势是恢复之后的状态与失效之前的 Namenode 保持高度一致性，不会出现数据丢失的情形；缺点就是同步所带来的带宽的消耗。

（2）通过从 Namenode 来实现当主 Namenode 失效之后的迅速切换，从而保证文件系统对外的服务能力不会中断。由于从 Namenode 运行于另一个节点之上，存在从 Namenode 保存的状态落后主 Namenode 当前状态的现象，导致的结果就是部分数据的丢失现象。

2.1.4 数据流

在客户端执行读取操作时，客户端和 HDFS 之间存在交互过程以及 NameNode 和各 DataNode 之间存在交互的数据流。

1．文件读取

HDFS 读取文件时时序图如图 2.2 所示：

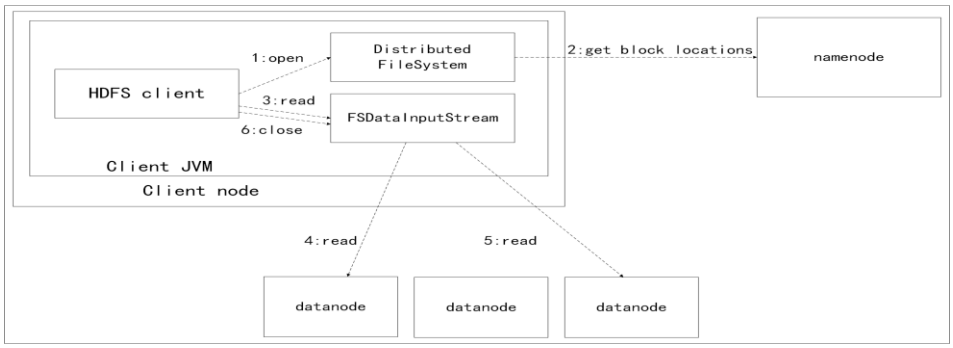

图 2.2 HDFS 读取文件

具体文件读取步骤如下所述：

（1）客户端创建 FileSystem 类的实例，打开目标文件通过创建对象的 open()方法实现。

（2）DistributedFileSystem 类为 FileSystem 类的子类，先前创建的 FileSystem 类的实例实际上真正创建的是 DistributedFileSystem 类的实例。由于客户端所在节点与 NameNode 所在节点并非同一节点，故需要 DistributedFileSystem 类实例通过远程过程调用（RPC）来访问 NameNode 获取目标文件起始数据块所在节点信息。由于 HDFS 中副本因子数目为 3，故集群中三个节点存放了文件起始数据块，因此，返回哪个副本所在节点信息需要 NameNode 作出决策，其决策思路如下：首先判断客户端节点是否为集群中的某个 DataNode 且存放有目标数据块，若是，则直接本地读取；若不是，则需要对存放数据块节点到客户端节点的距离进行排序，返回距离最小的 DataNode 信息。

（3）上述操作结束返回 FSDataInputStream 类的实例，该实例内部包含 DFSInputStream 类的实例，其主要的用途为文件定位，通过调用该实例的 read()方法实现与距离最近的 DataNode 的连接以及数据的传输，通过对 read()方法的反复调用实现目标数据块完整读取，该数据块读取完毕之后需要继续读取目标文件的后续数据块，该过程对于客户端来说是透明的。

（4）一旦文件的所有数据块读取完毕，则通过 FSDataInputStream 类的实例调用 close()方法实现连接断开，标志着整个文件读取的结束。

但是在上述步骤中没有考虑某些特殊情况，例如：连接至 DataNode 进行数据读取时发生异常情况，针对上述情况，往往采取的应对措施是：首先与 NameNode 进行通信，向其报告集群中的该 DataNode 发生故障并请求获取存放该数据块副本的次近的 DataNode 信息。NameNode 收到来自客户端的报告后标记该 DataNode 为故障节点，后续返回节点信息时不会返回该节点信息，避免其他客户端发生相同状况。客户端接收到来自 NameNode 的新的数据块所在 DataNode 信息之后，重复上述步骤，判断是否能够正常读取。对于可能存在的数据块损坏的情况通过校验来判断数据块的完整性。

2．文件写入

HDFS 写入文件时时序图如图 2.3 所示：

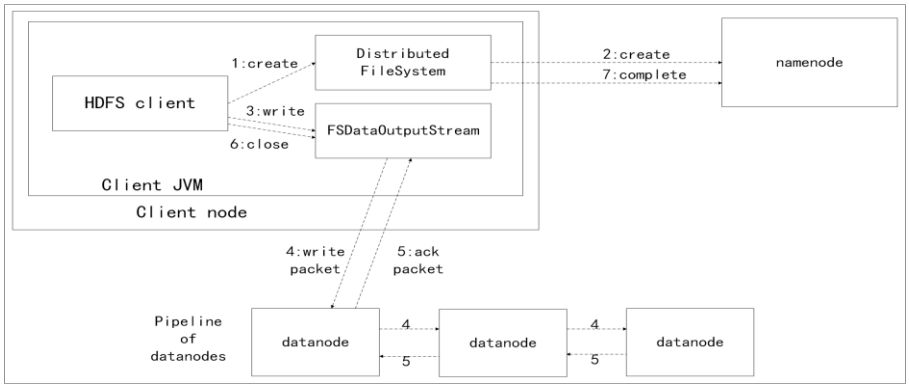

图 2.3 HDFS 写入文件

HDFS 写入文件时时序图如图 2.3 所示。具体文件读取步骤如下所述：

（1）客户端创建 DistributedFileSystem 类的实例，创建目标文件通过创建对象的 create()方法实现。

（2）由于客户端所在节点与 NameNode 所在节点并非同一节点，故需要 DistributedFileSystem 类实例通过远程过程调用（RPC）来访问 NameNode，在其 namespace 中新建文件。由于对用户来说存在读取权限的限制，因而 NameNode 需要判断用户是否具有写文件的权限以及 namespace 中是否已有用户想要写的文件。若验证通过，则在 namespace 中新建一条记录，否则抛出异常，告知用户原因。返回 FSDataOuputStream 类的实例，该实例内部包含 DFSOutputStream 类的实例，其主要的用途为文件定位。

（3）通过调用 DFSOutputStream 类的实例的 write()方法实现文件的写入操作。对文件进行写操作的过程中，首先需要做的是将文件分割为若干个数据块，按数据块进行写入，之后将数据块分割成若干个数据包，将其存放于实例的内部队列之中；与 NameNode 进行通信，获取存放该数据块的 DataNode 信息（返回集群中三个节点信息），对这三个节点进行数据块的写入，按照流水线的方式，客户端传送数据包至第一个 DataNode，第一个 DataNode 接收存储并将其传送至第二个 DataNode，第二个 DataNode 存储并将其传送至第三个 DataNode。由于网络传输过程中可能存在的异常情况，需要通过确认机制来判断数据包是否存储至流水线中的每一个节点，只有收到每一个节点的确认才可以从队列之中删除对应数据包。

（4）一旦文件的所有数据块写入完毕，则通过 FSDataOutputStream 类的实例调用 close()方法实现连接断开，标志着整个文件写入的结束。

对于文件写入，为了避免存在脏数据的情况发生，对用户可见的只有已经写完的文件而不是当前正在写的文件。其通过 sync()方法来实现，一旦写文件客户端调用 close()方法时，其内部包含的 sync()方法也会随之被调用，也就标志着文件写入的结束，此操作过后其它用户即可对该文件进行相应操作。

2.1.5 容错机制

由于 HDFS 集群中的机器数目庞大且性能较低，因此服务过程中出现故障是不可避免的情形，设计之初，完善的容错机制能够保证 HDFS 的持续性对外服务，对于其中可能出现的故障通常分为如下三类：

（1） 节点失效是指 NameNode 的单点失效（SPOF）[39] 以及 DataNode 宕机情形；

（2） 系统网络故障；

（3） 数据损坏。

针对上述可能出现的故障，HDFS 的容错机制给出的应对策略如下：

（1）对于 NameNode 的单点失效问题，通过配置活动-备用 NameNode 来应对活动 NameNode 失效时通过备用 NameNode 对外提供服务，这种方式不会让用户察觉到服务的中断；对于 DataNode 的宕机问题，首先是通过心跳机制来发现的，集群中的 DataNode 每隔 3s 会向 NameNode 发送心跳用于表明其存活性，一旦某个 DataNode 一段时间内（10 分钟）未发送心跳，则存在两种情况：其一为该 DataNode 已经宕机；其二为集群中的网络故障；此时需要 NameNode 向该 DataNode 发送消息，等其响应，若一定时间内未收到响应，需要通过一定次数的重试来验证，若始终未收到确认消息则可判定 DataNode 节点失效，此时 NameNode 为了保证集群系统中文件副本数目的一致性需要根据本地存储的该 DataNode 存放的文件副本列表来进行恢复操作，保证副本因子数目始终为 3。

（2）对于集群系统中可能发生的网络故障，则需要通过请求-确认-重传机制来判断，一旦确定数据中心内部发生网路故障则需要立即修复，以便后续对外提供服务。

（3）数据失效的可能因素是硬件设备出现问题，或者是由于系统软件上出现 bug，当然

也可能是数据传输过程中由于网络原因导致的错误。HDFS 拥有数据文件内部校验措施，为了提升效率，文件初始创建之时会对组成文件的每一个数据块进行校验和的计算并将其保存在 NameNode 之上方便后续数据块的校验。用户发起文件读取时，计算读取的数据块的校验和，与存储在 NameNode 上的值进行比对，若相同，文件没有损坏，读取成功；若不同，则存在损坏，需要从其它数据节点读取数据。

2.2 HDFS 副本管理技术

通常情况下，规模比较大的 Hadoop 集群通常包含多个数据中心，即使对于一些规模较小的集群来说通常也包含多个机架，为了保证数据的可用性，显然数据块的多个副本不能存放于同一数据节点。

2.2.1 默认副本存放策略

默认副本存放策略[40]的基本处理过程是（默认复制因子的数目为 3）：

（1）如若客户端本身是集群中的某个 DataNode，那么客户端所在节点即为第一个副本存放所在节点；否则，NameNode 通过随机算法选择集群中任一节点作为第一个副本存放所在节点。需要注意的是，NameNode 选择任一节点时会考虑节点当前负载情况以及近一阶段节点写入次数。

（2）第二个副本的存放需要通过 NameNode 随机选择与第一个副本不是同一个机架的其它机架上的节点当中。

（3）对于第三个副本，则需要存放在和第二个副本同一个机架但不同的节点当中。

（4）一旦选定副本存放位置，则系统会创建管线，将副本写入选定存放的 DataNode。

由于选择将其中两个副本放置于同一机架不同节点之上而不是将三个副本放置于三个不同机架的节点之上，减少了机架之间写入次数，因此降低了内部带宽的消耗，写入性能得到显著的提升；与此同时，考虑到集群中节点发生故障的可能性，选择将副本存放于不同机架之上，机架发生故障的概率要远远低于节点发生故障的概率，因此能够提供良好的稳定性与

负载均衡性能。确保了文件的高可靠以及高可用的同时降低写入操作所需的网络带宽的消耗。HDFS 默认副本存放策略如图 2.4 所示：

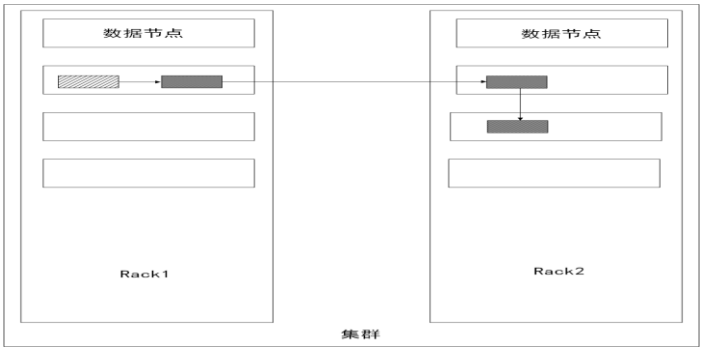

图 2.4 HDFS 默认副本存放策略

下面讨论副本存放位置的选择策略：

（1）选择一个本地节点

如若上传数据的客户端是 HDFS 集群中某个机架上的一个 DataNode，那么将其称为本地节点，优先考虑将数据写在本地节点之上。但是，写入之前首先需要获取节点当前的磁盘剩余空间大小，若剩余空间足够存放一个数据块，则将该数据块保存在客户端所属节点之上，此节点即为本地节点；如果不能满足上述条件即磁盘空间不足，对于这种情况，NameNode 通过选择客户端所在节点的同一机架上的某一节点作为副本存放的节点将其作为本地节点；最后，如果客户端不是 HDFS 集群中，则从整个集群中任一机架中的任一节点，作为副本存放的节点将其作为本地节点。其中，isGoodTarget()方法的调用来决定数据节点存放数据块的合适性。

（2）选择一个本地机架节点

本地机架节点的选择需要以（1）中所取节点作为参考，如果参考节点所在机架之上存在合适的 DataNode，则选择将其作为本地机架节点；如果未发现合适的 DataNode（机架之上的所有节点除参考节点之外剩余磁盘空间不足），那么需要考虑参考节点在步骤（1）中的选取是随机选取还是参考节点就是客户端所在节点或者是客户端所在机架之上的某一节点情况：对于随机选取的情况则需要重现选择本地节点，以选取的新本地节点作为参考点选择本地机架节点，重复上述过程，直至选择到合适的本地机架节点为止；对于后续两种情况则通过选择集群中任一机架之上的任一节点作为本地机架节点。

（3）选择一个远程机架节点

远程机架节点的选取依旧需要以（1）中所取节点作为参考，NameNode 随机选择集群中一个合适的 DataNode，此 DataNode 与参考点不在同一机架上；若集群中不存在满足上述要求的 DataNode，此时唯一的选项是选择与（1）中所取节点同一机架之上的任一节点作为远程机架节点。

（4）随机选择若干数据节点

NameNode 通过随机算法从集群中任意选择若干满足要求的 DataNode。

（5）优化数据传输路径

前文已经交代了数据块写入所采用的方式是流水线，那么，如何规划选定副本存放的 DataNode 的写入路径，来实现写入带宽的降低？NameNode 通过调用 ReplicationTargetChooser 类提供 getPipeline()方法来对选定的 DataNode 的写入顺序的排序，规划出最佳写入路径，按照选定的路径对 DataNode 进行数据块的流水线式写入。

副本复制[41]的基本处理方法是：

（1）NameNode 发现存在某个数据块的数目低于当前设置的副本因子或者某个节点或机架发生失效情况，发出重新备份的指令；

（2）NameNode 负责副本复制过程的调度，选择副本复制过程中需要使用到的目标节点与源节点；

（3）源节点接收到副本复制命令后对目标节点进行数据块的写入；

（4）数据块写入完毕之后，目标节点向源节点发送确认信号；

（5）源节点将复制完毕信号发送给名称节点，标志着副本复制过程的完结。

副本调整的基本处理方法是：

在集群负载不均衡或者有一批新的机器被添加到集群中时，集群需要通过 balancer 进程进行负载均衡操作，将负载高的节点中的副本移动至负载低的节点。

一般情况下，副本存放位置的选择策略中远程机架节点的选择具有随机性，未考虑节点目前负载数目、磁盘 I/O 效率以及与已选中本地机架节点之间的距离，导致远程机架节点副

本写入时内部带宽的消耗与节点间负载不均衡的情形，因此远程机架节点的选择是需要进一步改进的。

2.2.2 机架感知策略

为了避免机架失效情况下可能导致的数据丢失最终无法恢复，故数据块的所有副本需要存放于不同的机架之上。

机架感知策略[42]被 HDFS 用来判断 DataNode 当前的存活性以及用于提升访问效率。Hadoop 系统中的 NameNode 可以通过机架感知策略掌握每一个 DataNode 所在机架信息。显而易见的是，同一机架上的 DataNode 之间的数据传输速率远高于不同机架 DataNode 之间。文件读取时 NameNode 告知客户端每个块对应的最佳的 DataNode 就是依赖机架感知策略，通过缩短客户端与目标节点之间的距离，一方面能够提升数据读取的效率，另一方面也能减少内部带宽的消耗。

机架感知具体实现方法如下：任一 DataNode 在 HDFS 集群中的位置都是通过 dfs.network.script 参数来进行配置的，其指出 DataNode 的域名或 IP 地址到机架标识符之间的映射，据此，NameNode 通过加载该文件可以获取到集群中所有数据节点所属机架。换言之，NameNode 也就获取了集群中所有 DataNode 的网络拓扑结构。

HDFS 机架感知策略默认情况下是关闭的。因而，在这种情形下，对于节点的选择具有任意性，换言之，写数据时可能发生如下情形：将文件的第一个数据块（block1）写到机架 A（rackA）之上，随后在随机选择的情况下将第二个数据块（block2）写入机架 B（rackB）之上，这一阶段的写需要跨机架，远超过同一机架上不同节点之间的写流量的消耗。最后，对于第三个数据块（block3）在随机选择的情况下又被写回了机架 A（rackA）。上述过程，产生了两次跨机架间写的流量，造成数据中心内部带宽的急剧消耗，成为限制服务能力的重要因素之一。

默认情形时，NameNode 启动时日志如下所示：

2018-01-26 18:36:57,423 INFO org.apache.hadoop.net.NetworkTopology: Adding a new node: /default-rack/ 192.168.117.82:50010

/default-rack 表示任意一个机架的 IP 对应的都是默认机架，标志着 HDFS 默认情况下是关闭机架感知策略的。

若需要开启 HDFS 机架感知策略，首先需要定位集群中的 NameNode 所在节点，其次定位该节点路径/home/bigdata/apps/hadoop/etc/hadoop 下的配置文件 core-site.xml，最后通过在配置文件尾部添加下述语句来实现:

```
<property>
    <name> default_rack_start.script.file.name</name>
    <value>/home/bigdata/apps/hadoop/etc/hadoop/ default_rack_start.sh</value>
</property>
```

机架感知策略启动脚本（default_rack_start.sh）的主要功能是实现从 DataNode 的 IP 地址到该节点所在机架的映射，该脚本接收的参数为 DataNode 的 IP 地址，输出为该 DataNode 所在的机架。当 NameNode 启动之后，需要加载配置文件，判断用户是否进行相应设置，若判定用户进行相应设定，则表明需要开启机架感知策略，NameNode 根据脚本中的相应设定查找目标路径下的脚本文件。此后，一旦 DataNode 向 NameNode 发送心跳，NameNode 则会执行脚本文件，将得到的映射关系保存至内存当中。

default_rack_start.sh 脚本的简单实现如下：

```
#!/bin/bash
HADOOP_CONF=/home/bigdata/apps/hadoop/etc/hadoop
while [ $# -gt 0 ]; do    //determine weather the number of arguments is greater than zero
  nodeArg=$1    //$1 means DataNoded's IP
  exec<${HADOOP_CONF}/topology.data //the contents of topology.data as stdin
  result=""
  while read line; do
    ar=( $line )
    if [ "${ar[0]}" = "$nodeArg" ]||[ "${ar[1]}" = "$nodeArg" ]; then
      result="${ar[2]}"
    fi
  done
  shift    //shift the parameters left
```

```
if [ -z "$result" ] ; then    //if $result is null, return true; else, return false
   echo -n "/default-rack"
else
   echo -n "$result"
fi
done
```

通过上述配置之后，NameNode 启动时日志如下所示：

2018-01-26 19:01:25,272 INFO org.apache.hadoop.net.NetworkTopology: Adding a new node: /dc1/rack3/ 192.168.117.85:50010

上述语句则表明机架感知策略被启用。

HDFS 内部网络可以当作树状结构，任意两个节点之间的距离为这两个节点到最近公共祖先的距离之和。按照场景，可得如下的分类情形：

（1）同一节点上的进程；

（2）同一机架上的不同节点；

（3）同一数据中心不同机架上的节点；

（4）不同数据中心的节点。

那么给出定义如下：dA 表示 A 数据中心，rA 表示 A 机架，nA 表示 A 节点；上述情形可以按照如下方式描述其间距离：

（1）distance(/dA/rA/nA, /dA/rA/nA) = 0 表示同一节点上的进程；

（2）distance(/dA/rA/nA, /dA/rA/nB) = 2 表示同一机架上的不同节点上的进程；

（3）distance(/dA/rA/nA, /dA/rB/nA) = 4 表示同一数据中心不同机架上的节点；

（4）distance(/dA/rA/nA, /dB/rA/nA) = 6 表示不同数据中心的节点。

根据上述讨论可以发现，节点间的距离与其所属机架、机架所在数据中心有关，不同数据中心的节点之间的距离最长而同一数据中心且同一机架之上的节点之间的距离最短。机架感知策略的存在实现了节点所在机架位置的获取，依据节点所属机架信息则可以计算集群中任意两个节点之间的距离，节点间的距离是后续改进默认副本存放策略中选端机架选择的重要因素之一。

2.3 其他分布式文件系统的副本存放策略

GFS 中默认的数据块副本数目为 3，配置文件中对应系统默认数据块副本的数目的值可以按照系统的需求进行相应改动。

2.3.1 GFS 副本存放策略

GFS 副本存放策略[43]主要实现两个目标：一个是数据可靠性和可用性，另一个是网络带宽利用率。

通常情况下，GFS 集群中包含的数以千计的 Chunk 服务器分布在不同的机架之上，而客户端访问这些服务器时，客户端与服务器一般不在同一机架之上，由于不同机架之间的节点间的访问通常需要通过一个或者多个交换机，因此，如果只是简单地将副本存放于不同节点之上无法达到上述的两个目标，故副本最终会被放置到不同机架之上。那么，即使发生节点甚至机架失效的现象，也可以保证数据的可靠性与可用性。与此同时，由于副本被放置于不同机架之上，对于数据访问请求，则可以根据客户端与副本所在节点的位置远近选择最优节点，提升内部网络带宽的利用率。

GFS 选择 Chunk 服务器时需要考虑的因素有：硬盘使用率较低，最近一段时间没有发生过副本创建活动，以及不同机架之上。

2.3.2 TFS 副本存放策略

组成 TFS 集群的节点同样为廉价机器，因此发生故障的可能性比较高，为了解决节点失效带来的 Block 丢失的问题，TFS 通过增加 Block 副本来解决 [44]。默认情况下，TFS 系统中的每一个 Block 存在两个副本，为了提升其容错性，选择不同网段之上的不同 DataServer 进行存放。

对于每一个 Block 写入来说，只有当其数目达到 TFS 系统设定的副本因子才能算作写入完毕。对于 TFS 系统中发生的由于节点失效以及节点的磁盘损毁情形，通过对副本进行复制并将其写入适当节点作为应对措施，保证任何时刻任一 Block 数目始终大于或等于副本因子

的值。

对于读取/写入过程中由于网络因素可能导致的 Block 数据失效的情形，TFS 系统的应对措施是对读取/写入的每一个 Block 进行校验，通过比对前后两次校验值来判断是否出现 Block 数据失效，若前后校验值不同，则需要从其它存放该 Block 的 DataServer 中进行读取，与此同时对于失效的 Block 需要进行相应的恢复操作即通过复制覆盖该 DataServer 中存放的失效 Block。

GFS 选择副本存放节点时考虑节点目前磁盘使用率以及近一阶段副本创建活动数目，TFS 对于失效数据块的删除操作采用覆写的实现方法，对于 HDFS 默认副本存放策略的改进都起到借鉴的作用，对于远端机架节点的选择考虑节点当前负载，对于失效副本，通过覆写而不是删除操作实现节点 I/O 操作次数的减少，进而提升对外提供服务的能力。

2.4 本章小结

本章论述了分布式文件系统 HDFS 中的数据流与可用性，详细描述 HDFS 默认副本存放策略的处理流程，指出了其中存在的问题，提出了改进思路。同时讨论了分布式文件系统 GFS 和 TFS 的副本存放策略，分析其中存在的利弊，为后续的副本存放策略改进奠定基础。

第三章 HDFS 副本存放策略的改进

HDFS 的默认副本存放策略为任意选择远端机架节点存放的策略[45-46]，虽然在一定程度上考虑了节点的状态信息，比如：磁盘使用情况，可用空间，负载信息等，其中负载信息的获取是通过对该节点的读取与写入的连接数进行大致的估算得到的。

3.1 默认副本存放策略存在的问题

需要注意的是，HDFS 的默认副本存放策略对负载信息的获取是在选定存储节点之后才发生的，所以副本最终存放的节点是无法被控制的。其存在的问题主要体现在如下几个方面：

（1）HDFS 选取第一个副本存放节点时遵循本地节点的原则，若存在用户通过某一客户端持续写入数据，恰巧该客户端是系统中的一个 DataNode，则按照上述原则该节点会被持续写入，造成节点间的负载不均衡。

（2）使用不同机架来存放同一数据块的不同副本来保证数据的高可用性，如果集群包含多个机架，那么可以实现数据块的高可用性以及实现数据的均衡分布，但一旦集群只包含一个机架，那么随之而来的问题是数据块无法实现均匀分布以及数据块的高可用性则无法被保证。

（3）节点当前的负载数目没有纳入节点选择的考虑范围之内，可能发生的情况是负载较高的节点会被持续的写入，使得集群中不同节点负载出现两种极端的现象（某些节点负载很高，其它节点负载很低），一旦负载高的节点失效，则会带来的问题是进行数据块恢复时会造成集群内部网络带宽的巨大消耗。

（4）如果存在对数据块有较大的读请求，那么可以通过增加副本数目来提升读性能，但是对于超过三个的副本块，后续的每一个副本存放节点的选择都是随机的，这就会造成上述（3）的问题。

对于默认副本存放策略，HDFS 内部通过 Balancer 守护进程来应对上述问题对。Balancer 通过 RPC 访问 NameNode 获取集群中数据块分布情况以及各 DataNode 当前存储现状，通过

移动数据块，实现降低负载较高的节点当前的数据块数目（删除该节点上已移动副本），从而提升负载较低的节点的当前数据块数目（将冗余节点上的副本移动至此节点），最终的目的是实现集群中数据块数目均匀分布。Balancer 守护进程可以说在一定程度上缓解了默认副本存放策略存在的问题，但问题没有得到彻底的解决，其原因主要有：

（1）对数据块的移动调整是滞后的，当发现数据块的存储存在不均匀的现象之后才通过移动数据块解决分布不均匀的状况。

（2）集群内部带宽资源的不合理消耗，与其事后纠正前期存在的问题，不如在进行远端机架节点选择时就进行优化，从而避免后期的纠正，减少因为数据块移动而产生的内部带宽的消耗。

3.2 改进的副本存放策略

通过上述分析已经明确了默认副本存放策略存在的问题，改进的副本存放策略所需要实现的目标如下：

（1）可靠性：通过多份副本存放于不同机架之上来实现数据块的高可靠性，发生节点失效情况时能够实现数据块的恢复，保证系统中数据块的数目不低于最低要求；

（2）可用性：通过将副本的份数设为多份，增强数据块的可用性，与此同时也能实现数据块读性能的提升；

（3）均衡性：这是改进副本存放策略主要提升的方面，通过选择合理的远端机架目标节点，使得各节点数据块负载的均衡性，避免后期负载不均衡导致的数据块移动。

依据上述分析发现的问题以及改进副本存放策略的目标，需解决的关键问题应该考虑如何选择存放在副本的远端机架，而不是在存放完毕之后再去考虑补救之法，以此来优化 HDFS 默认副本存放策略，以达到进一步提升 HDFS 的服务能力。

之所以将副本中存放在远端机架节点之上，是为了提升数据块的高可靠以及高可用。一旦第一副本存放的本地节发生节点失效时，能够通过远端机架上的副本进行数据块的恢复工作；当客户端进行数据块的读取工作时能够选取最靠近客户端的副本存放节点。本文的改进

副本存放策略着重考虑解决如下问题：

（1）HDFS 假设集群中的所有 DataNode 拥有相同的性能，但这在现实情况下是不存在的，各个节点之间不可避免的存在差异性，对于数据节点来说重要的是其读写性能，因此忽略其它差异性重点关注各个节点的磁盘 I/O 速率，这是衡量集群中 DataNode 服务能力的一个重要指标。

（2）针对跨机架节点选择时，需要考虑的因素必然有副本存放所在的本地节点与副本存放的远端机架上的节点之间的距离。显而易见，当两者之间的距离越短，数据恢复时所需要消耗的集群内部带宽也就越少，数据恢复的速度也就越快，因此，节点之间的距离也是衡量 DataNode 服务能力的一个重要指标。

（3）集群中每一个 DataNode 的磁盘空间都是有限的，对于默认副本存放策略来说，由于节点选择时的随机性，故副本可能过于集中的分布于几个节点之上，造成该节点副本存放过多，影响后续该节点在集群中的服务能力（不断地并发写入与并发读取）。因此，节点当前的负载情况是衡量 DataNode 服务能力的又一重要指标。

（4）系统中通常会存在那些经常失效的或发生错误的副本，针对这一情况，需要实现这类副本的快速恢复。因此，采取如下方式，设定某一阀值，通过阶段性的分析获取失效和错误的次数超过该值的副本，调整（1）、（2）、（3）中三个指标的系数，完成对于此类副本的迅速恢复。这一方式通过动态调整副本存放的位置来提升副本的快速恢复能力。

用户上传文件主要流程如下：用户通过客户端进行文件上传操作时，NameNode 对集群中的可用节点进行匹配，选择匹配度最高的节点响应用户，用户随后将文件传送到目标节点：

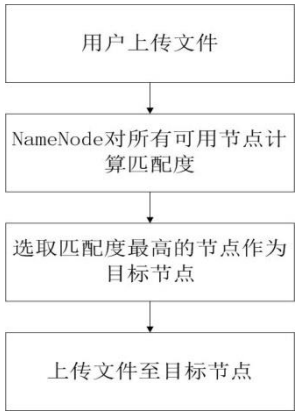

图 3.1 用户上传文件流程图

本文改进的副本存放策略的处理流程如下：用户通过客户端进行文件上传操作时，NameNode 对集群中的可用节点进行匹配，并改进其中的匹配度计算方法：

（1）任意选择特定数值的不同机架上的 DataNode 作为目标节点作为选择；

（2）获取所有目标节点的磁盘 I/O 效率；

（3）获取所有目标节点到本地节点的距离值；

（4）获取目标节点当前存放的副本的数目；

按照匹配度计算公式，依据上述获取的三个数值计算出每一个节点的匹配度，选取匹配度最高的节点作为副本存放的远端机架的目标节点。

通过定时任务获取阶段性的副本的失效次数与错误次数之和，一旦发现上述结果超过指定阀值，需要依照上述提供策略对于副本存放节点进行重新选择，与先前不同的地方是，对于这类副本，需要调整上述三个因素的比例系数，用于提升热点失效与错误副本恢复的速度。

图 3.2 给出了改进副本存放策略的处理流程图：

图 3.2 改进副本存放策略思想流程图

3.3 节点匹配度评价方法

如上所述，副本存放节点选择之时，改进的策略会按照事先获取的节点的磁盘 I/O 效率，当前负载信息，节点间的距离来给出该目标节点的匹配度，该值被作为目标节点选择的指标，具体计算方式如下：

（1）磁盘 I/O 效率

对于任一 DataNode 的磁盘 I/O 效率来说，需要考虑的无外乎读取效率与写入效率，记磁盘 I/O 效率为 V，得出如下公式：$V = \alpha 1 VRead + \alpha 2 VWrite$。

（2）节点负载信息

节点负载信息可以通过 NameNode 获取，记节点负载为 C。

（3）节点间的距离

节点间的距离可以通过 NetworkTopology 类中 getDistance()获取，记节点间的距离为 L。

如上所述，对于每一个目标节点，改进的策略会依据节点的磁盘 I/O 效率，节点当前负载信息，节点间的距离来给出该目标节点的匹配度，将该值作为目标节点选择的依据，具体计算方式如下：

$$P（i）= \alpha 1 \frac{1}{C(i)} + \alpha 2 \frac{1}{L(i)} + \alpha 3 V(i)$$

式中：C(i)为编号为 i 的目标节点当前的负载数目，该值与节点的匹配度成反比；L(i)为编号为 i 的目标节点到当前的网络距离，该值与节点的匹配度成反比；V(i)为编号为 i 的目标节点的磁盘 I/O 效率，该值与节点的匹配度成正比；α1、α2、α3 为比例系数，需要根据不同的匹配度评估时刻进行合适的指定，分别针对用户上传文件至集群时和针对副本错误与失效次数超过阀值时的节点选择。对于用户上传文件时 α1、α2、α3 分别指定为 30%、30%、40%；对于副本错误与失效次数超过阀值时的节点选择，需要提高 α2 的值来实现对于副本的快速恢复，降低 α1 的值，此时节点当前负载对于节点匹配度的影响因子较小，故 α1、α2、α3 分别指定为 15%、45%、40%。

节点匹配度评价方法执行流程如图 3.3 所示：

图 3.3 节点匹配度评价方法执行流程图

3.4 算法描述

基于上面的设计思想，本文改进的副本存放策略的算法处理流程如下：

（1）当有新的文件需要上传时；

（2）对于第一个副本节点选择，按照本地节点选择的原则，若客户端自身是集群中的某个 DataNode，只要磁盘有足够空间存放该数据块则将它存放于客户端所在节点；若不满足上述条件，则由 NameNode 通过 NetworkTopology 类中 getDistance()获取距离客户端较近的几个节点作为目标节点，应用上述匹配度计算方法，计算每个节点的匹配度，选取目标节点中匹配度最高的节点作为本地节点；

（3）以步骤（2）中选取的存放节点作为参考点，进行远端机架节点的选择。通过 NameNode 获取已匹配节点列表中节点的数目，若该数目小于指定值 K，则转向（4）；否则转向（5）；

（4）随机选取节点，若选取节点不在已匹配节点列表中并且该节点与已匹配节点列表中任一节点不在同一机架，则将该节点添加至待匹配节点列表；

（5）依据匹配度计算方法，计算待匹配节点列表中每一个待匹配节点的匹配度，将计算结果记录至已匹配节点列表，清空待匹配节点列表；

（6）对已匹配节点列表按照匹配度进行排序；

（7）返回已匹配节点列表中匹配度最高的节点，作为副本存放的远端机架的目标节点。

用户上传文件时远端机架副本存放节点选择策略的算法流程如图 3.4 所示：

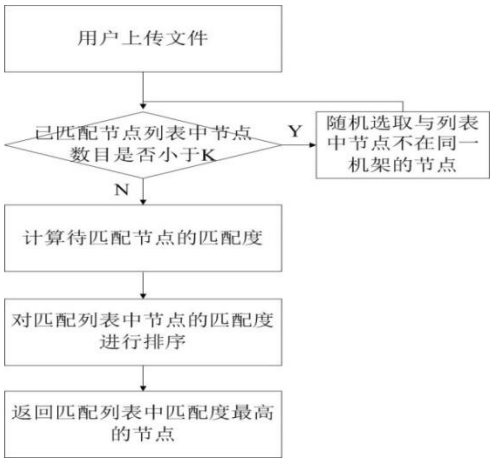

图 3.4 用户上传文件时远端机架副本存放节点选择策略

上述方式描述了对于新的文件需要上传时的改进的副本存放策略，对于副本错误与失效次数之和超过指定阀值时，相对应的处理策略如下：

（1）当副本错误与失效次数之和超过指定阀值时，分为如下两种情况：①同一机架中某一节点上的副本发生上述情况；②远端机架中的副本发生上述情况；对于情形①来说只需通过同一机架之间的写复制即可实现副本恢复；对于情形②来说，需要首先执行步骤（2）；

（2）以“幸存”副本中当前读写压力较小的节点作为参考点，进行远端机架节点的选择。通过 NameNode 获取已匹配节点列表中节点的数目，若该数目小于指定值 K，则转向（3）；否则转向（4）；

（3）随机选取节点，若选取节点不在已匹配节点列表中并且该节点与已匹配节点列表中任一节点不在同一机架，则将该节点添加至待匹配节点列表；

（4）依据匹配度计算方法，计算待匹配节点列表中每一个待匹配节点的匹配度，将计算结果记录至已匹配节点列表，清空待匹配节点列表；

（5）对已匹配节点列表按照匹配度进行排序；

（6）返回已匹配节点列表中匹配度最高的节点，作为副本存放的远端机架的目标节点。

副本错误与失效次数之和超过指定阀值时副本恢复流程如图 3.5 所示：

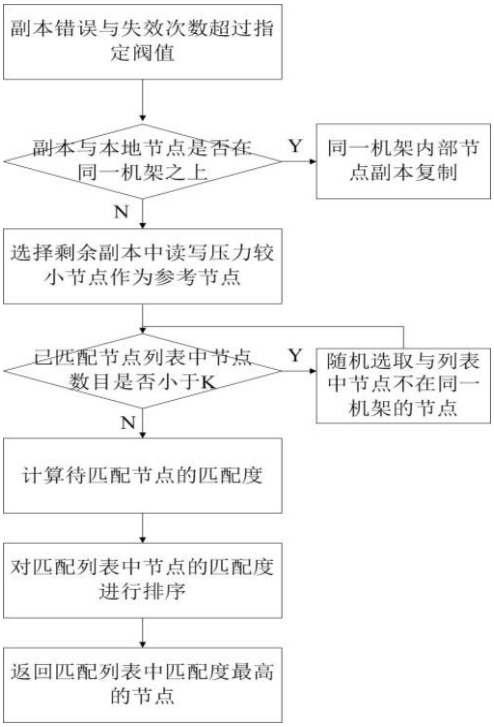

图 3.5 副本恢复时存放节点选择策略

3.5 实现方法

Hadoop 默认副本存放策略进行节点选择的处理步骤如下：

（1）通过默认数据块存放策略类 BlockPlacementPolicyDefault 的 chooseTarget()方法进行节点选择；

（2）通过调用 NetworkTopolcgy 类的 contains()方法来判断客户端是否是集群中的一个 DataNode，若客户端是集群中的一个 DataNode，则通过 chooseLocalStorge()方法选择客户端作为第一个节点；否则通过调用 chooseRandom()方法随机选择集群中的任一机架中的一个节点，通过 chooseLocalRack()方法在本地机架上随机选择一个节点作为第一个节点；一旦选定了节点，那么如何判断该节点是否合适呢？通过 isGoodTarget()方法来判断，判断的点主要有节点剩余容量是否足够，节点当前的负载状况，通过上述判断后才能确定为副本存放的节点；

（3）通过调用 chooseRemoteRack()方法在远程机架上随机选择一个节点作为第二个节点，使用 NetworkTopolcgy 类的 isOnSameRack()方法来判断该节点是否与前一个节点是否在同一个机架上，若不在同一机架上，则将该节点作为第二个节点，同样需要通过 isGoodTarget()方法来判断是否符合要求；

（4）通过调用 chooseLocalRack()方法在第二个节点的本地机架上随机选择一个节点作为第三个节点，通过 isGoodTarget()方法来判断是否符合要求。

根据 HDFS 默认副本存放策略，依照本文提出的算法改进思路，对应的主要处理程序需做如下修改：

```
public class ImprovedBlockPlacementPolicy extends BlockPlacementPolicy {
    /*NetworkTopology 用来表示 hadoop 集群的网络拓扑结构*/
    private NetworkTopology clusMap;
    /*其余私有成员变量未标注*/
    public DataNodeDesriptor chooseTarget(
        //文件系统相关信息
        FSInodeInfo fsInodeInfo,
        //副本文件的份数
```

```
            int countReplicas
            DataNodeDescriptor descriptor,
            List<DataNodeDescriptor> targetNodes,
            long blockNum) {
               //省略部分重复代码
               double[] calc = new double[randomNum];
               for (int k=0; k<randomNum; k++) {
                  //进行节点选择
                  DataNodeDescriptor dataNodeDes = choseRandom(...);
                  //计算节点匹配度
                  calc[k] = match(dataNodeDes);
               }
               //获取匹配度最高的节点
               double max_calc = sort(calc);
               return choose_max(max_calc);
         }
      }
```

3.6 本章小结

本章在分析了默认副本存放策略存在的问题的基础上，提出改进副本存放策略所需要实现的目标，具体说明了改进副本存放策略的实现方法，并给出了集群中 DataNode 计算匹配度的公式，实现改进副本存放策略的算法，最后通过改进原有的实现方法来完成改进副本存放策略对默认副本存放策略的替换。

第四章 基于可信第三方的 Kerberos 审计问责方案

如果云存储服务商本身是不可信的，或者可能存在云存储服务商因为自身利益侵害用户数据，或者由于机械及系统故障导致数据错误或丢失现象时云存储服务商推卸责任，那又如何实现与存储数据的安全性呢？通过提出基于可信第三方的 Kerberos 审计问责方案是一种有效的解决途径。

4.1 云存储数据安全性现状

当前，已有许多云存储服务商采用访问控制表（ACL）来实现对用户访问权限的控制，其中比较出名的就是 Amazon。在 Amazon S3 当中，用户存放于云端的数据被抽象为桶的形式，系统规定不同的桶需要有不同的命名，以桶为数据存放的基本单位。假定在桶 jayzhou 下存放文件 video/bunengshuodemimi.mp4，那么用户访问该资源的方式是通过在浏览器的地址栏键入 jayzhou.s3.amazonaws.com/video/bunengshuodemimi.mp4。至于该用户是否拥有权限访问该资源，则是通过访问控制表中该桶对应的可访问用户表中是否包含该用户来实现的，绝大多数的云存储服务商都是通过上述方式实现用户访问权限控制的。

上述方式成功地解决了未拥有对应文件权限的用户无法访问该文件的问题，但对云存储服务商本身的不可信或者由于机械与系统故障导致的数据错误或丢失等问题，如何保障存储数据的安全性呢？通过增加基于可信第三方的 Kerberos 审计问责方案，对 Kerberos 的身份认证协议作出相应改进，解决其中存在的口令猜测攻击等问题，同时通过对 Ticket 的修改来实现审计问责的功能，这是一个有效的技术途径。

当然，如果不能保证云存储服务商是可信的，存储于云端的所有数据必须通过需要加密等手段来从根本上解决云端数据的安全性问题。

4.2 Kerberos 简介

Kerberos 身份认证协议是由麻省理工学院（Massachusetts Institute of Technology，MIT）

提出的，设计之初的目的是用于开放网络环境中的身份认证，也可保证数据的完整及其保密。Kerberos Distributed Center（KDC）作为 Kerberos Authentication 中 Client 和 Server 信任的第三方，在身份认证过程中发挥至关重要的作用。

Kerberos 认证系统出了客户端（Client）之外，包含的服务器如下：

（1）认证服务器（Authenticator Server 简称 AS）：用户登录时对其身份进行验证；

（2）授予许可证服务器（Ticket Granting Service 简称 TGS）：发放身份证明许可证；

（3）服务端（Server）：客户端实际请求资源的对象；

Kerberos 认证流程过程中涉及的处理策略如下：

（1）Sclient-server：Client 和 Server 会话期间使用的公钥。同理，Skdc-client 与 Skdc-server 也就不难理解，分别代表 KDC 和 Client 会话期间使用的公钥、KDC 和 Server 会话期间使用的公钥；

（2）Authenticator：Client 证明自身的依据。通常情况下包含用户名、用户密码等身份信息以及当前时间的时间戳，时间戳存在的原因是验证双方时间同步下若时间戳距离验证方当前时间超过某一阈值（一般情况下为 5 分钟）则认为该验证消息被破译的可能性很大，因此终止此次验证服务；

（3）TGT(Ticket Granting Ticket)：TGT 与具体 Server 无关，Client 只有获取到 TGT 之后方可向 KDC 申请访问某个具体 Server 的 Ticket；

（4）Ticket：Server 向 Client 提供服务的先决条件；

（5）Master Key：KDC、Client、Server 拥有各自的公钥，通过对密码进行哈希运算生成，由于哈希的不可逆性，只需保存好自身的密码即可保证其安全性；

Kerberos 认证系统执行流程如下：

（1）Client 向 KDC 的 AS 发送 TGT 的请求消息，请求信息中包含：用户名、用户密码、TGS 名称、时间戳等；通过 Client 的 Master Key 对请求消息中的用户密码和时间戳进行加密处理；

（2）KDC 端的 AS 接收到来自 Client 的请求消息之后，通过用户名查询本地数据库，获

取 Client 端的 Master Key，对加密信息进行解密，获取用户密码和时间戳，首先验证用户密码是否一致从而判定用户的合法性，其次判断时间戳是否在可接受范围之内，若不在可接受范围之内则认为消息被拦截破译拒绝进行服务；否则，生成响应消息，包含如下内容：Client 端的 Master Key 加密的 Skdc-client，KDC 端的 Master Key 加密的 TGT，TGT 中包含 Skdc-client、Client 信息以及 TGT 到期时间。将响应消息发送至 Client；

（3）Client 接收到 AS 响应消息之后，通过自身的 Master Key 解密获取 Skdc-client，构造向 TGS 发送的用于获取 Ticket 的请求消息，请求消息包含如下内容：TGT、Client 的信息（使用 Skdc-client 加密）、客户端用户名、服务端用户名。向 KDC 端发送获取 Ticket 的请求消息之前，请求 Server 端获取其缓存的 TGT；

（4）Server 接收到客户端请求之后，判断自身是否缓存 TGT，若未缓存，则仿造（1），（2）两步获取属于 Server 端的 TGT 以及使用 Server 端的 Master Key 加密的 Skdc-server，返回 TGT 至客户端；否则，直接返回缓存的 TGT 至客户端；

（5）Client 接收到 Server 响应的 TGT 之后，将步骤（3）中构造的获取 Ticket 的请求消息与服务端响应的 TGT 一并发送至 TGS；

（6） TGS 接收到 Client 端的请求消息之后，通过自身的 Master Key 首先解密客户端从 AS 获取的 TGT，从而获取到 Skdc-client、Client 信息以及 TGT 到期时间。通过 Skdc-client 解密用户信息，通过用户信息比对判断用户的合法性。

验证通过后，解密 Server 响应 Client 的 TGT，获取 Skdc-server。向 Client 发送响应消息，其中包含：使用 Client 的 Master Key 加密的 Sserver-client、使用 Skdc-server 加密的 Ticket 信息，Ticket 中包含 Sserver-client、用户信息以及 Ticket 到期时间；

（7）Client 接收到 TGS 响应消息之后，通过自身的 Master Key 解密获取 Sserver-client，向 Server 发送请求消息，请求消息包含如下内容：Ticket 以及 Client 信息（使用 Sserver-client 加密）；

（8）Server 接收到 Client 的请求消息之后，通过 Skdc-server 解密 Ticket 获取其中包含的 Sserver-client、用户信息以及 Ticket 到期时间；通过 Sserver-client 解密 Client 信息，进行比对，

验证用户的合法性。验证通过，同意用户访问请求；否则，拒绝服务。

4.3 基于可信第三方的 Kerberos 审计问责方案

一般来讲，可信第三方问责系统通常需要实现如下三方面的功能：

（1）记录用户对云端数据所做操作：由于假想云端不可信，因此该记录不能仅存放于云端，可能存在云端篡改用户所做操作的行为，所以需要将该操作记录副本存放于可信第三方；

（2）一旦发现数据安全问题，可信第三方需要能够通过云端存放记录与可信第三方存放记录的比对来判定责任的归属，快速准确的找到事故的责任方，维护用户以及云存储服务商的权益不受侵犯；

（3）可信第三方出具的记录具有不可抵赖性，用户以及云存储服务商都需要在可信第三方生成的用户操作记录进行信息确认，一旦确认无误，则需要添加用户、云存储服务商以及可信第三方的签名至记录，实现记录的不可抵赖性。

4.3.1 改进方案基础架构设计

通过对 Kerberos 认证流程的叙述可以发现其在认证方面具有高可靠性，因此本文的改进方案选择依赖其基础架构，通过引入可信第三方的概念，修改 KDC 端的设计、传输数据的加密方式以及适度简化认证流程的来实现审计问责的功能。

本文设计的可信第三方 KDC，具有如下基础模块以及新增功能：

（1）基础信息表：user_info_table，包含字段 user_id,user_name, user_password,master_key，其中 user_name 设置为唯一键，master_key 通过 user_password 进行哈希运算生成，需要注意的是用户不止是客户端也包括服务端；user_key_info_table，包含字段 user_id,user_name, user_key，其中 user_ key 是通过加密模块为客户端生成的非对称加密的公钥；file_info_table，包含字段 file_id, file_name, file_version, file_key, file_obj_uuid, file_hash_code，is_exist 其中 file_key 是当前版本文件的解密秘钥，file_hash_code 是当前版本文件计算得出的哈希值，主要用于与云端存储文件的哈希值进行比对从而判断云端文件是否发生修改，每次对文件进行

操作都会在 file_info_table 表中新增一条最新版本文件信息记录，相对于前一版本记录发生变化的是 file_version, file_obj_uuid, file_hash_code 以及针对当前版本的新 file_key, 其中 file_obj_uuid 为唯一键，通过 UUID 生成的文件的唯一标识，is_exist 用于标识文件是否新建成功；

（2）加密模块：负责非对称加密密钥对以及对称加密密钥的生成。

（3）访问控制功能：通过基于角色的访问控制列表来实现对用户请求资源的权限验证；每当用户上传文件至云端时，KDC 端需要在访问控制列表中添加相应记录便于后续的权限控制，上传文件用户拥有添加其他用户对上传文件允许进行哪些操作的权限。

（4）传输数据加密方式：借助可信第三方的目的就是假想云端无法确保存储数据的安全性，对于上传文件所做的第一步是用户上传文件至 KDC 加密，通过 KDC 端的非对称加密模块生成密钥对，通过服务端的 Master Key 先做对称加密操作，其后再用秘钥对中的公钥对其进行非对称加密，需要注意的是，由于本方案中文件具有版本号，且不同版本的文件其非对称加密的秘钥不同，因此 KDC 端需要存放文件当前版本到解密秘钥的映射，为了后续云端对文件进行操作时的文件解密；上述方式将 Kerberos 原先的对称加密和时间戳方式修改为对称加密与非对称加密的结合，提升了网络传输过程中的安全性。

用户（User）：其功能相对简单，主要包括：上传文件、对文件权限范围内的操作、发现数据安全性问题时向 KDC 发起审计问责请求。

云服务提供商：分配存储空间、执行用户文件操作请求、用户污蔑数据安全问题时向可信第三方发起审计问责请求，存放用户文件操作记录通过数据库的表实现记为 ticket_table，包含字段 file_obj_uuid, user_id, timestamp, manipulate_type, manipulate_res，以 file_obj_uuid, user_id, timestamp 作为主键标识用户对文件的一次操作记录。

Ticket 是数据安全问题发生之后 KDC 进行审计问责的依据，也是用户和云存储服务商不可抵赖的凭证，因而 Ticket 的设计至关重要。KDC 端的 Ticket 信息由 ticket_table 表负责信息存放。

ticket_table 包含 ticket_id, file_obj_uuid, user_id, timestamp, manipulate_type,

manipulate_res, user_signature, server_signature, cloud_signature。其中 file_obj_uuid, user_id, timestamp 作为唯一键方便后续与云端存放的操作记录信息进行比对。manipulate_type 包含 CREATE，DELETE， UPDATE，READ，ADUIT；manipulate_res 包含 CREATE SUCCESS/FAIL，DELETE SUCCESS/FAIL， UPDATE SUCCESS/FAIL， READ SUCCESS/FAIL， AUDIT SUCCESS/FAIL。通过 user_signature，server_signature，cloud_signature 这三个签名可以实现责任判定之后的不可抵赖性。

4.3.2 方案执行流程设计

KDC 的加密模块为 KDC，Client 和 Server 生成非对称加密秘钥对，通过 KDC 端存放的 Client 的 Master Key 对 Client 的秘钥以及 KDC 和 Server 的公钥进行加密之后发送, Server 处理方式同上，确保每一方都存放己方的秘钥以及另外两方的公钥，便于后续文件传输的加密操作。

Client 向 Server 请求 Sserver-client，若服务端已缓存 Sserver-client，则发送至 Client；否则，通过 Client 请求 KDC 的加密模块为 Client, Server 生成对称加密秘钥（Sserver-client），通过 Client 端的公钥对响应消息进行非对称加密处理之后使用 Client 端的 Master Key 进行对称加密处理，将响应消息发送至 Client；Server 端通过类似方式获取响应消息。

Client 向 KDC 发送 Ticket 请求消息，请求消息中包含：客户端用户名、客户端密码、服务端用户名、请求操作文件名称、请求操作类型以及时间戳等；通过 KDC 端的公钥对请求消息中的客户端密码和时间戳信息进行非对称加密处理之后使用 Client 的 Master Key 进行对称加密处理。

KDC 接收到来自 Client 的 Ticket 请求消息之后，通过消息中的用户名获取 Client 端的 Master Key 及其公钥信息，对加密信息进行解密，验证信息的合法性。验证无法通过，拒绝进行服务；否则，生成 Ticket 响应消息，包含如下内容： Client 信息、请求操作文件名称、请求操作类型以及 Ticket 到期时间。通过 Server 端的公钥对响应消息进行非对称加密处理之后使用 Server 端的 Master Key 进行对称加密处理，将响应消息发送至 Client。此时，KDC 端

需要生成 Client 此次操作的保存于 KDC 端的 Ticket。

Client 接收到来自 KDC 端的 Ticket 响应消息之后，向 Server 发送请求消息，请求消息中包含：Ticket 以及 Client 信息，其中 Client 信息通过 Server 端的公钥对其进行非对称加密处理之后使用 Sserver-client 进行对称加密处理，将请求消息发送至 Server。

Server 接收到 Client 的请求消息之后，通过 Server 端的 Master Key 以及己方私钥解密 Ticket 获取其中包含的 Client 信息、请求操作文件名称、请求操作类型以及 Ticket 到期时间；通过 Sserver-client 以及己方私钥解密获取 Client 信息，进行比对，验证用户的合法性。验证通过，同意用户访问请求；否则，拒绝服务。

Server 执行请求操作完毕之后，需要存放一条 Client 此次请求操作的 Ticket 记录，响应 Client 告知此次操作完毕，将己方签名发送至 KDC 端，用于告知 KDC 此次用户请求操作完成，生成对应 Ticket 记录。

Client 收到来自 Server 端的响应消息之后，需要在线判断云端数据是否与所要进行的操作一致，若一致，将己方签名发送至 KDC 端，用于 KDC 端 Ticket 记录字段信息的填充；若不一致，则告知 KDC 端此次请求操作未成功。

KDC 端收到 Server 以及 Client 的用户签名之后，若 Client 以及 Server 都响应操作成功则在 Ticket 记录中添加上述用户签名字段；若 Client 以及 Server 响应操作结果不一致则需要 KDC 端进行审计问责流程。

KDC 端存放云端文件对应最新版本映射表的原因是为了解决可能存在的用户并发修改云端数据的现象：假定在用户 A 修改云端文件 F 之前用户 B 已对文件 F 作出相应修改，而用户 A 请求修改云端文件 F 的版本还是之前一个版本，若 KDC 端未存放对应文件最新版本号可能导致用户 A 在不知情的情况下修改了用户 B 已经做过修改的文件而云端也没有将这个事情通知用户 A，而是任由他作出相应的改动，因此 KDC 端需要存放云端文件对应最新版本映射表。

综上所述，针对每次用户对云端数据的操作都会产生相对应的 Ticket 记录，该记录中包含用户签名、云端签名以及 KDC 端签名，分别存放于 KDC 端以及云端之上。

本文改进方案的执行流程如图 4.1 所示：

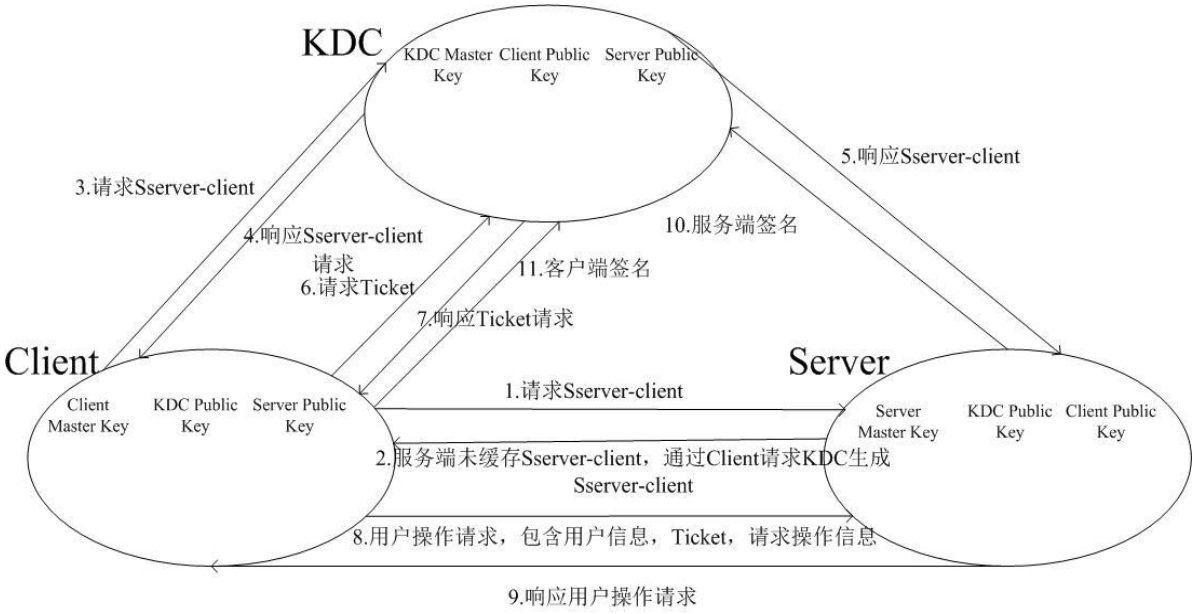

图 4.1 改进方案的执行流程图

4.3.3 用户校验流程

用户校验流程包含如下两个步骤：

（1）Client 向 KDC 发送 Ticket 请求消息时，请求消息中包含客户端用户名以及客户端密码等。对于客户端密码通过 KDC 端的公钥进行非对称解密处理之后使用 Client 的 Master Key 进行对称解密处理；此步骤验证了用户名与密码的匹配性，若不匹配则无法解密获取请求消息中的加密信息。

（2） Client 向 Server 发送的请求消息中包含：Ticket 以及 Client 信息，通过解密获取 Ticket 中存放的 Client 信息与解密后的 Client 信息进行比对，判定用户的合法性。

用户校验流程图如图 4.2 所示：

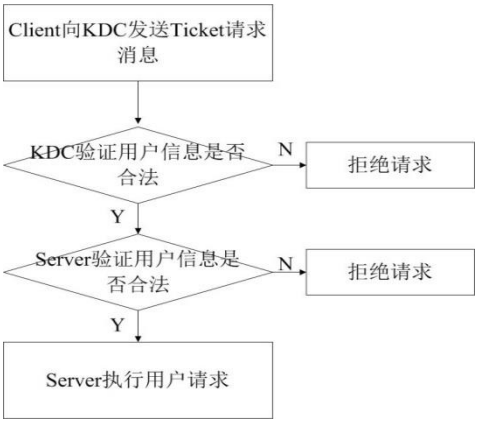

图 4.2 用户校验流程

4.4 数据交互

（1）创建文件

用户上传文件至 Server 时，需要向 KDC 发送 Ticket 请求，请求消息中包含的字段如表 4.1 所示：

表 4.1 上传文件时客户端请求字段

user_name	用户名
user_password	用户密码
server_name	云端名称
Timestamp	上传文件请求时间戳
manipulate_type	CREATE
file_name	文件名称

KDC 端接收到来自用户的 Ticket 请求消息之后，获取用户的 Master Key 及其公钥信息，对加密信息进行解密，验证用户信息的合法性。验证无法通过，拒绝进行服务；通过访问控制表判断用户是否拥有创建文件权限，若无，则拒绝服务；否则，生成 Ticket 响应消息，包含如下内容： Client 信息、请求操作文件名称、请求操作类型以及 Ticket 到期时间。通过 KDC 端的公钥对响应消息进行非对称加密处理之后使用 KDC 端的 Master Key 进行对称加密处理，将响应消息发送至 Client。

KDC 端需要生成用于问责依据的 Ticket，需要填充的字段包括：ticket_id, file_obj_uuid, user_id, timestamp, maniplate_type，其中 file_obj_uuid 通过 KDC 生成用于唯一标识该文件版本。发送至 Client 的 Ticket 响应消息中包含的字段如表 4.2 所示：

表 4.2 上传文件时 KDC 响应 Ticket 请求字段

user_name	用户名称
file_name	文件名称
manipulate_type	CREATE
Timestamp	上传文件请求时间戳
ticket_deadline	Ticket 到期时间

客户端接收到 KDC 的 Ticket 响应消息之后，将上传文件、Ticket 响应消息以及用户信息发送至 KDC，其中 Ticket 响应消息无需处理，用户信息以及上传文件需要通过 KDC 端的公钥进行非对称加密处理之后使用 Client 的 Master Key 进行对称加密处理。

KDC 端接收到客户端的请求消息之后，解密 Ticket 响应消息、用户信息，比对判断用户的合法性，验证通过之后解密上传文件，通过密钥模块生成非对称加密密钥通过 file_info_table

中新增一条记录用于保存上传文件信息（上传文件，故 file_version=1,file_key 为非对称加密私钥，计算文件哈希值 file_hash_code）。上传文件通过生成的非对称加密的公钥来进行加密，KDC 端向 Server 发送请求消息，请求消息中包含：上传文件，file_obj_uuid, user_id, timestamp, manipulate_type。通过 Server 端的公钥进行非对称加密处理之后使用 Server 的 Master Key 进行对称加密处理，请求消息包含如下字段如表 4.3 所示：

表 4.3 上传文件时 KDC 对云端请求

user_name	用户名称
user_id	用户 Id
file_name	文件名称
file_obj_uuid	文件对象唯一标识
Timestamp	上传文件请求时间戳
manipulate_type	操作类型

Server 接收到 KDC 的请求消息之后，完成文件的存储后对请求消息进行解密，根据上述字段生成存放于 Server 的 Ticket 记录为后续问责做准备。Server 需要对 KDC 作出响应，响应消息包含如下字段如表 4.4 所示：

表 4.4 上传文件时 Server 响应 KDC

user_id	用户 Id
file_obj_uuid	文件对象唯一标识
Timestamp	上传文件请求时间戳
manipulate_type	CREATE
manipulate_res	CREATE SUCCESS
cloud_signature	云端签名

为了告知 Client 文件上传完毕，Server 需要对 Client 作出响应，响应消息包含如下字段如表 4.5 所示：

表 4.5 上传文件时 Server 响应 Client

user_name	用户名称
file_name	文件名称
Timestamp	上传文件请求时间戳
manipulate_type	CREATE
manipulate_res	CREATE SUCCESS

Client 接收到 Server 的响应消息之后需要判断上传文件操作的结果是否与 Server 的响应消息中 manipulate_res 描述一致，若不一致则需要发起审计问责；若一致则需要响应 KDC，响应消息包含如下字段如表 4.6 所示：

表 4.6 上传文件时 Client 响应 KDC

user_id	用户 Id
file_obj_uuid	文件对象唯一标识
Timestamp	上传文件请求时间戳
manipulate_type	CREATE
manipulate_res	CREATE SUCCESS

KDC 接收到 Client 的响应消息之后，若 Client 与 Server 响应消息中的 manipulate_res 字段一致，file_info_table 中对应文件的 is_exist 字段设置为 1，生成完整的 Ticket 信息，Ticket 包含如下字段如表 4.7 所示：

表 4.7 上传文件时的 Ticket

ticket_id	Ticket Id
file_obj_uuid	文件对象 UUID
user_id	用户 Id
Timestamp	上传文件请求时间戳
manipulate_type	CREATE
manipulate_res	CREATE SUCCESS
user_signature	用户签名
server_signature	云端签名
kdc_signature	可信第三方签名

至此，文件上传操作流程结束。

（2）读取文件

Client 读取 Server 存储的文件时，向 KDC 发送 Ticket 请求，请求消息中包含的字段如表 4.8 所示：

表 4.8 读取文件时 Client 的 Ticket 请求字段

user_name	用户名
user_password	用户密码
server_name	云端名称
Timestamp	读文件请求时间戳
manipulate_type	READ
file_name	文件名称

Client 信息以及读权限校验过程不再赘述与文件上传时一致。

KDC 端生成 Ticket 响应消息，包含如下内容： Client 信息、请求操作文件名称、请求操作类型以及 Ticket 到期时间。通过 Server 端的公钥对响应消息进行非对称加密处理之后使用 Server 端的 Master Key 进行对称加密处理，将响应消息发送至 Client。KDC 端需要生成用于问责依据的 Ticket，需要填充的字段包括：ticket_id, file_obj_uuid(根据 file_name 查询 file_info_table 可知), user_id, timestamp, maniplate_type。发送至客户端的 Ticket 响应消息中包含的字段如表 4.9 所示：

表 4.9 读取文件时 KDC 响应 Ticket 请求字段

user_name	用户名称
user_id	用户 Id
file_name	文件名称
file_obj_uuid	文件对象 UUID
manipulate_type	READ
Timestamp	读文件请求时间戳
ticket_deadline	Ticket 到期时间

Client 向 Server 请求 Sserver-client(server 与 client 会话期间的对称加密密钥)，若服务端已缓存 Sserver-client，则发送至 Client；否则，通过 Client 请求 KDC 的加密模块为 Client, Server 生成 Sserver-client，通过 Client 端的公钥对响应消息进行非对称加密处理之后使用 Client 端的 Master Key 进行对称加密处理，将响应消息发送至 Client；Server 端通过类似方式获取响应消息。

客户端接收到 KDC 的 Ticket 响应消息之后，将 Ticket 响应消息以及用户信息发送至 Server, 其中 Ticket 响应消息无需处理，用户信息需要通过 Server 端的公钥进行非对称加密处理之后使用 Sserver-client 进行对称加密处理。请求消息中的用户信息包含的字段如表 4.10 所示：

表 4.10 读取文件时 Client 对 Server 请求

user_name	用户名
file_name	文件名称
Timestamp	读文件请求时间戳
manipulate_type	READ

Server 接收到 Client 的请求消息之后，通过对比解密之后的用户信息对比验证用户的合法性，若信息不一致则拒绝服务；否则 Server 通过 Ticket 消息中的 file_name 字段请求 KDC 返回该文件的私钥以及操作完毕之后新的非对称加密公钥信息，获取文件的私钥之后解密文件执行用户查询请求，响应 Client 请求，告知 KDC 此次 Client 请求操作的响应结果之后按照

新的公钥对文件加密，根据 Ticket 中的字段信息生成此次用户操作的 Ticket 记录。Server 需要对 KDC 作出响应，响应消息包含的字段如表 4.11 所示：

表 4.11 读取文件时 Server 响应 KDC

user_id	用户 Id
file_obj_uuid	文件对象唯一标识
Timestamp	读文件请求时间戳
manipulate_type	READ
manipulate_res	READ SUCCESS
cloud_signature	云端签名

Client 查看响应消息是否与预期一致，若不一致则需要发起审计问责；否则需要响应 KDC，响应消息包含的字段如表 4.12 所示：

表 4.12 读取文件时 Client 响应 KDC

user_id	用户 Id
file_obj_uuid	文件对象唯一标识
Timestamp	读文件请求时间戳
manipulate_type	READ
manipulate_res	READ SUCCESS

KDC 接收到 Client 的响应消息之后，若 Client 与 Server 响应消息中的 manipulate_res 字段一致，则需要更新 file_info_table 中对应文件的 file_key，生成完整的 Ticket 信息，Ticket 包含的字段如表 4.13 所示：

表 4.13 读取文件时的 Ticket

ticket_id	Ticket Id
file_obj_uuid	文件对象 UUID
user_id	用户 Id
Timestamp	读文件请求时间戳
manipulate_type	READ
manipulate_res	READ SUCCESS
user_signature	用户签名
server_signature	云端签名
kdc_signature	可信第三方签名

至此，文件读取操作流程结束。

（3）更新文件

Client 更新 Server 存储的文件时，向 KDC 发送 Ticket 请求，请求消息中包含的字段如表 4.14 所示：

表 4.14 更新文件时 Client 的 Ticket 请求字段

user_name	用户名
user_password	用户密码
server_name	云端名称
Timestamp	更新件请求时间戳
manipulate_type	UPDATE
file_name	文件名称

Client 信息以及权限校验过程不再赘述与上述过程一致。

KDC 端生成 Ticket 响应消息，包含如下内容： Client 信息、请求操作文件名称、请求操作类型以及 Ticket 到期时间。通过 Server 端的公钥对响应消息进行非对称加密处理之后使用 Server 端的 Master Key 进行对称加密处理，将响应消息发送至 Client。生成用于问责依据的 Ticket，需要填充的字段包括：ticket_id, file_obj_uuid(根据 UUID 算法生成), user_id, timestamp, maniplate_type。发送至客户端的 Ticket 响应消息中包含的字段如表 4.15 所示：

表 4.15 更新文件时 KDC 响应 Ticket 请求字段

user_name	用户名称
user_id	用户 Id
file_name	文件名称
file_obj_uuid	文件对象 UUID
manipulate_type	UPDATE
Timestamp	更新文件请求时间戳
ticket_deadline	Ticket 到期时间

Client 向 Server 请求 Sserver-client 方式与读文件请求时类似不再赘述。

Client 接收到 KDC 的 Ticket 响应消息之后，将 Ticket 响应消息、更新文件内容以及用户信息发送至 Server,其中 Ticket 响应消息无需处理，更新文件内容以及用户信息需要通过 Server 端的公钥进行非对称加密处理之后使用 Sserver-client 进行对称加密处理。请求消息中的用户信息包含的字段如表 4.16 所示：

表 4.16 更新文件时 Client 对 Server 请求

user_name	用户名
file_name	文件名
Timestamp	更新文件请求时间戳

manipulate_type	UPDATE

Server 接收到 Client 的请求消息之后，通过对比解密之后的用户信息对比验证用户的合法性，若信息不一致则拒绝服务；否则 Server 通过 file_name 字段请求 KDC 返回该文件的私钥以及操作完毕之后新的非对称加密公钥信息，获取文件的私钥之后解密文件按照更新文件内容执行用户更新请求，计算文件哈希值 file_hash_code，响应 Client 请求，告知 KDC 此次 Client 请求操作的响应结果之后按照新的公钥对文件加密，根据 Ticket 中的字段信息生成此次用户操作的 Ticket 记录。Server 需要对 KDC 作出响应，响应消息包含的字段如表 4.17 所示：

表 4.17 更新文件时 Server 响应 KDC

user_id	用户 Id
file_obj_uuid	文件对象唯一标识
file_hash_code	文件哈希值
Timestamp	更新文件请求时间戳
manipulate_type	UPDATE
manipulate_res	UPDATE SUCCESS
cloud_signature	云端签名

Client 查看响应消息是否与预期一致，若不一致则需要发起审计问责；否则需要响应 KDC，响应消息包含的字段如表 4.18 所示：

表 4.18 更新文件时 Client 响应 KDC

user_id	用户 Id
file_obj_uuid	文件对象唯一标识
Timestamp	更新文件请求时间戳
manipulate_type	UPDATE
manipulate_res	UPDATE SUCCESS

KDC 接收到 Client 的响应消息之后，若 Client 与 Server 响应消息中的 manipulate_res 字段一致，则需要在 file_info_table 中新增一条记录用于对应文件当前最新版本，生成完整的 Ticket 信息，Ticket 包含的字段如表 4.19 所示：

表 4.19 更新文件时的 Ticket

ticket_id	Ticket Id
file_obj_uuid	文件对象 UUID
user_id	用户 Id
Timestamp	更新文件请求时间戳

manipulate_type	UPDATE
manipulate_res	UPDATE SUCCESS
user_signature	用户签名
server_signature	云端签名
kdc_signature	可信第三方签名

至此，文件更新操作流程结束。

（4）删除文件

Client 删除 Server 存储的文件时，向 KDC 发送 Ticket 请求，请求消息中包含的字段如表 4.20 所示：

表 4.20 删除文件时 Client 的 Ticket 请求字段

user_name	用户名
user_password	用户密码
server_name	云端名称
Timestamp	删除文件请求时间戳
manipulate_type	DELETE
file_name	文件名称

Client 信息以及删除权限校验过程不再赘述与文件上传时一致。

KDC 端生成 Ticket 响应消息，包含如下内容： Client 信息、请求操作文件名称、请求操作类型以及 Ticket 到期时间。通过 Server 端的公钥对响应消息进行非对称加密处理之后使用 Server 端的 Master Key 进行对称加密处理，将响应消息发送至 Client。KDC 端需要生成用于问责依据的 Ticket，需要填充的字段包括：ticket_id, file_obj_uuid(根据 file_name 查询 file_info_table 可知), user_id, timestamp, maniplate_type。发送至客户端的 Ticket 响应消息中包含的字段如表 4.21 所示：

表 4.21 删除文件时 KDC 响应 Ticket 请求字段

user_name	用户名称
user_id	用户 Id
file_name	文件名称
file_obj_uuid	文件对象 UUID
manipulate_type	DELETE
Timestamp	删除文件请求时间戳
ticket_deadline	Ticket 到期时间

Client 向 Server 请求 Sserver-client 的过程与读取文件时一致不再赘述。

客户端接收到KDC的Ticket响应消息之后，将Ticket响应消息以及用户信息发送至Server，其中 Ticket 响应消息无需处理，用户信息需要通过 Server 端的公钥进行非对称加密处理之后使用 Sserver-client 进行对称加密处理。请求消息中的用户信息包含的字段如表 4.22 所示：

表 4.22 删除文件时 Client 对 Server 请求

user_name	用户名
file_name	文件名
Timestamp	删除文件请求时间戳
manipulate_type	DELETE

Server 接收到 Client 的请求消息之后，通过对比解密之后的用户信息对比验证用户的合法性，若信息不一致则拒绝服务；删除指定文件之后，响应 Client 请求，告知 KDC 此次 Client 请求操作的响应结果，根据 Ticket 中的字段信息生成此次用户操作的 Ticket 记录。Server 需要对 KDC 作出响应，响应消息包含的字段如表 4.23 所示：

表 4.23 删除文件时 Server 响应 KDC

user_id	用户 Id
file_obj_uuid	文件对象唯一标识
Timestamp	删除文件请求时间戳
manipulate_type	DELETE
manipulate_res	DELETE SUCCESS
cloud_signature	云端签名

Client 查看响应消息是否与预期一致，若不一致则需要发起审计问责；否则需要响应 KDC，响应消息包含的字段如表 4.24 所示：

表 4.24 删除文件时 Client 响应 KDC

user_id	用户 Id
file_obj_uuid	文件对象唯一标识
Timestamp	删除文件请求时间戳
manipulate_type	DELETE
manipulate_res	DELETE SUCCESS

KDC 接收到 Client 的响应消息之后，若 Client 与 Server 响应消息中的 manipulate_res 字段一致，则需要生成完整的 Ticket 信息，Ticket 包含的字段如表 4.25 所示：

表 4.25 删除文件时的 Ticket

ticket_id	Ticket Id
file_obj_uuid	文件对象 UUID

user_id	用户 Id
Timestamp	删除文件请求时间戳
manipulate_type	DELETE
manipulate_res	DELETE SUCCESS
user_signature	用户签名
server_signature	云端签名
kdc_signature	可信第三方签名

至此，文件删除操作流程结束。

4.5 审计与问责

一旦 Client 发现 Server 存放的文件存在私自修改甚至删除或 Server 认为 Client 存在污蔑行为或多用户修改致使第一次更新丢失时，都可以向 KDC 发起审计请求。以 Client 发起审计请求为例，请求消息包含的字段如表 4.26 所示：

表 4.26 审计时的请求消息

user_name	用户名
user_password	密码
file_name	文件名
Timestamp	审计申请时间戳
manipulate_type	AUDIT

KDC 接收到来自 Client 的审计请求消息之后进行如下处理：

（1）验证用户信息合法性；

（2）通过访问控制列表判断用户是否具有发起审计文件的权限，若用户未拥有，则响应 Client 告知其拒绝审计的原因；否则，发起审计过程；

（3）发送 file_name 的解密秘钥至 Server 请求其计算对应文件的哈希值，将结果响应至 KDC；

（4）KDC 接收到响应消息之后判断 Server 计算所得文件哈希值与 file_info_table 中对应文件的最新版本的 file_hash_code 是否一致，若不一致，则判定 Server 存在擅自修改的现象；若一致，则需要通过存放于 Server 以及 KDC 的 Ticket 比对来进行问责；

（5）通过 file_name 获取 file_info_table 中该文件最新版本的 file_obj_uuid，由于 file_obj_uuid 的唯一性可以获取对应该字段的存放于 Server 以及 KDC 的 Ticket；

（6）若 Ticket 信息一致且 Ticket 记录操作与操作结果与文件当前信息一致，则判定为 Client 责任；否则判定为 Server 责任。

4.6 安全性分析

表 4.27 列出了本文设计的基于可信第三方 Kerberos 审计问责方案面临安全问题时对应的审计方法。

表 4.27 面临安全问题时对应的审计方法

安全问题	审计方法
Server 执行 Client 请求操作失败	Client 由响应可知
Client 否认对 Server 存放文件的操作	KDC 审计
Server 否认 Client 对文件的操作	KDC 审计
非法的 Client 操作	KDC 审计

（1）Server 执行 Client 请求操作失败：Client 的请求消息经过 KDC 以及 Server 验证之后由 Server 执行请求，若 Server 执行请求失败则需要通过响应消息告知 Client 以及 KDC 此次请求操作失败，KDC 以及 Server 对此次 Client 请求操作生成的 Ticket 也会记录操作结果，作为后续审计的依据。

（2）Client 否认对 Server 存放文件的操作：Server 向 KDC 发起审计请求，通过 KDC 以及 Server 存放的 Ticket 对比来审计 Client 是否进行过相应的操作，由于 Ticket 上具有 Client，Server 以及 KDC 的签名，因此如果发现 Client 的确进行相应操作，则判定为 Client 责任；否则判定为 Server 的责任。

（3）Server 否认 Client 对文件的操作：Client 向 KDC 发起审计请求，通过 KDC 以及 Server 存放的 Ticket 对比来审计 Client 是否进行过相应的操作，由于 Ticket 上具有 Client，Server 以及 KDC 的签名，因此如果发现 Client 的确进行相应操作，则判定为 Client 责任；否则判定为 Server 的责任。

（4）非法的 Client 操作：通过 KDC 存放的访问控制列表对用户请求类型进行权限判断，若该 Client 未具有请求类型的权限则拒绝此次请求；否则通过用户此次请求。

4.7 本章小结

本章基于 Kerberos 的认证流程，引入了第三方审计问责机制，设计了一种改进的基于 KDC 的问责方案，并具体说明了该改进方案的执行流程，详细描述了用户认证以及数据交互的过

程。当数据安全性问题发生时，通过审计与问责的流程，可以实现云端数据安全性以及审计问责的能力。

第五章 实验结果与分析

本章主要针对改进后的 HDFS 副本存放策略进行性能测试，通过搭建 Hadoop 集群环境，通过测试数据的对比分析来验证改进的副本存放策略所带来的性能优化。

5.1 实验环境

5.1.1 硬件环境

实验集群中包含 9 个节点，其中 1 个节点作为 NameNode，剩余 8 个节点作为 DataNode。将 8 个 DataNode 分布于 4 个机架之上，每个节点拥有相同的配置，双核 CPU，内存大小为 2GB，硬盘大小为 256GB，操作系统采用的是 ubuntu14.04（64 位）。

5.1.2 软件环境

由于需要修改 HDFS 默认副本存放策略，故实验需要如下两个实验环境：

（1） Hadoop 源码编译环境：对于修改 HDFS 默认副本存放策略的实现的需求，需要做的是对 Hadoop 源码下的子工程 hadoop-hdfs-project 中的副本选择源码实现作出相应的改动，改动之后，需要重现编译源码，将编译生成的 Jar 包上传至 Hadoop 集群运行环境进行相应的替换。主要涉及的包括如下：Homebrew 1.1.9（Mac OS 平台下的软件包管理工具，拥有安装、卸载、更新、查看、搜索等很多实用的功能），Maven 3.3.8，CMake 3.7.1（跨平台的编译工具），Mac os X 10.12.3（操作系统），JDK 1.7.0（JDK 版本）。

（2） Hadoop 源码运行环境如下：Hadoop 2.7.2（Hadoop 版本），JDK 1.8.0（JDK 版本），SSH protocol version 2，Ubuntu 14.04（操作系统），Vim 7.2。

5.2 环境搭建

5.2.1 源码编译

源码编译所需要实现的是对 Hadoop 源码下的子工程 hadoop-hdfs-project 中的副本选择源码实现做出相应的改动，替换为改进后的副本存放策略，重现编译源码，将编译生成的 Jar 包上传至 Hadoop 集群运行环境进行相应的替换。具体步骤如下所示：

（1）下载并安装 Hadoop 源码编译环境中所列出的软件；

（2）将修改后的 Hadoop 源码全部复制到指定新建的目录，然后进入该目录的第一层级。

（3）清理编译环境，通过使用 Maven 指令实现：mvn clean；用于减少编译过程中可能出现的不可预知的错误；

（4）输入编译命令，指定相应的参数：mvn package -Pdist, native -Dskip Tests -Dztar。初次编译耗时较长，直至出现 BUILD SUCCESS 表明源码编译成功；

（5）上传编译好的 Jar 至 Hadoop 集群之上。由于是对 Hadoop 默认副本存放策略作出改进，故需要修改的是子工程 hadoop-hdfs-project，按照上述命令，改进副本存放策略编译之后存放于 Hadoop-2.7.2-src/hadoop-hdfs-project/hadoop-hdfs/target 下的 hadoop-hdfs-2.7.2.jar 文件之中，需要将这个文件上传至集群 Hadoop 的安装目录 hadoop-2.7.2/share/hadoop/hdfs，之后重启 Hadoop 集群，那么改进副本存放策略则能生效。

5.2.2 Hadoop 集群环境

Hadoop 集群运行环境的具体搭建步骤如下所示：

（1）在 Hadoop 官网下载安装包 Hadoop 2.7.2.tar，将其解压至指定的目录之下，解压命令如下：tar -xvf ./hadoop-2.7.2.tar /usr/software；

（2）在 Oracle 官网下载安装包 jdk-8u161-linux-i586.tar.gz，将其解压至指定的目录之下，解压命令如下：tar -xvf ./jdk-8u161-linux-i586.tar.gz /usr/software；接下来需要做的是配置环境变量$JAVA_HOME，需要添加如下内容至 profile 文件尾部：

export JAVA_HOME=/usr/software/jdk1.8.0_161;

export PATH=$PATH:$JAVA_HOME/bin;

进入/usr/software/hadoop-2.7.2/etc/hadoop 目录，配置 Hadoop 运行所需的参数。

首先需要做的就是通过配置 core-site.xml 文件来开启 Hadoop 的机架感知策略，具体配置如下所述：

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://10.253.43.112:9000</value>
    </property>
    <property>
        <name>topology.script.file.name</name>
        <value>/usr/software/hadoop-2.7.2/topology.sh</value>
    </property>
</configuration>
```

其次需要配置的是默认副本的数目以及 NameNode 与 DataNode 的数据存放位置，可以通过 hdfs-site.xml 进行如下配置：

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>3</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/usr/software/hadoop-2.7.2/dfs/name</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/usr/software/hadoop-2.7.2/dfs/data</value>
    </property>
</configuration>
```

接下来需要在/usr/software/hadoop-2.7.2/etc/hadoop 目录下面创建文件 slave，其中记录的

是 Hadoop 集群中所有 DataNode 的 IP 地址；

集群中其余所有节点按照上述步骤进行相应的配置。

最后通过安装目录/usr/software/hadoop-2.7.2/etc/bin 目录下的 start-all.sh 来启动 Hadoop 集群。同时还需要验证 Hadoop 集群是否搭建成功，通过在 Master 节点如数 jps 命令，如果进程 Nmaenode，SecondaryNamenode 以及 ResourceManager 运行正常，则 Master 节点运行正常；在 Slave 节点输入 jps 命令，若进程 BootStrap，NodeManager 以及 Datanode 运行正常，则 Slave 节点运行正常。Master 以及 Slave 运行正常表明整个 Hadoop 集群运行正常，即表明 Hadoop 集群搭建成功。

5.3 实验结果分析

实验集群包含 1 个 NameNode 以及 8 个 DataNode，其中将这 8 个节点置于 4 个机架之上（机架分别命名为 A、B、C、D），每个机架包含 2 个 DataNode（每个 DataNode 对应一台机器）。

为了规避特殊性，保证选取提交 DataNode 的随机性，实验中，通过随机算法选中机架 C 中的某个节点作为提交数据的客户端。本次实验累计选取 3000 个大小相同的数据块，由 Hadoop 默认的副本存放策略可知，对于每一个数据块来说只有一个数据块副本存放在本地机架的节点之上，故另外 3 个机架（机架 A、机架 B、机架 D）总共需要存放 6000 的数据块，本地机架（机架 C）则需要存放 3000 个数据块，默认实验初始时每个机架的每个 DataNode 上存放的数据块数目为 0。

图 5.1 为默认副本存放策略下数据块在机架中的分布情况，显而易见，机架 A、机架 B、机架 D 中数据块数目分布不均匀，按照前文所述，与机架 C 相近的节点存放的数据块数目远大于距离较远的节点所存放的数据块的数目。随之而来的问题就是当发生数据块错误或者节点失效情况之后，由于数据块恢复而产生内部带宽的巨大消耗，同时，由于存在单个节点存放数据过多的现象，因此当频繁读取数据块时，对该节点的读写性能会是一个考验。

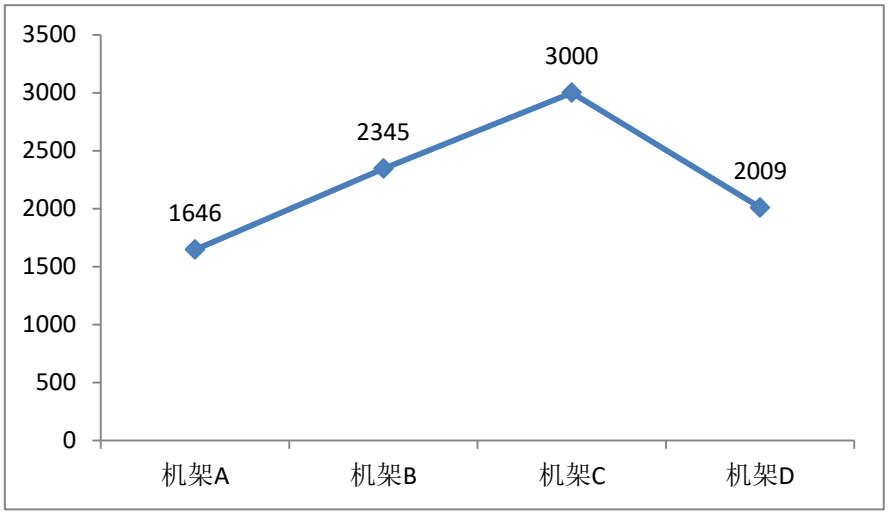

图 5.1 默认副本存放策略下数据块机架分布

图 5.2 为改进副本存放策略下数据块在机架中的分布情况，很明显的可以观察出各机架包含的数据块数目较为接近，成功的解决了默认副本存放策略所带来的数据块存放负载不均衡的现象，与此同时，所能够带来的好处降低了对数据块读写时内部带宽的消耗，由于数据块分布均匀，故不会因为频繁的读写需求对某个节点产生巨大压力。

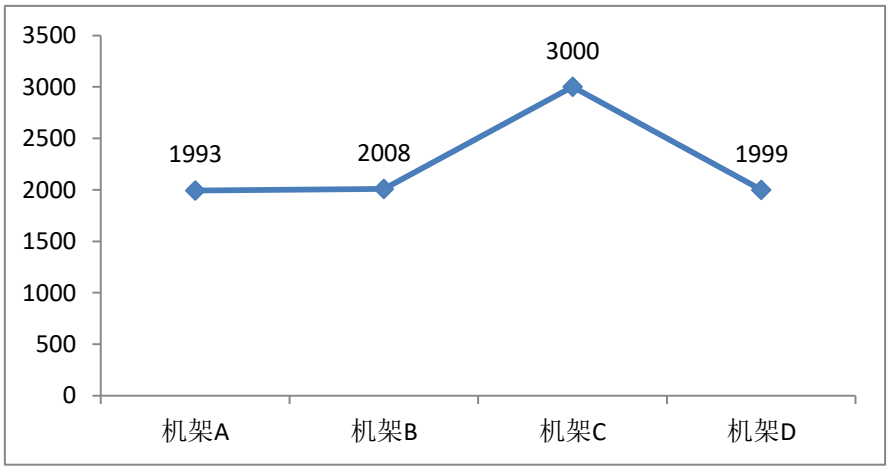

图 5.2 改进副本存放策略下数据块机架分布

对于改进副本存放策略，指定系数 0.3、0.3、0.4，目标节点的数目固定为 5，通过改进算法计算这 5 个节点的匹配度，选取其中匹配度最高的作为副本存放的节点。

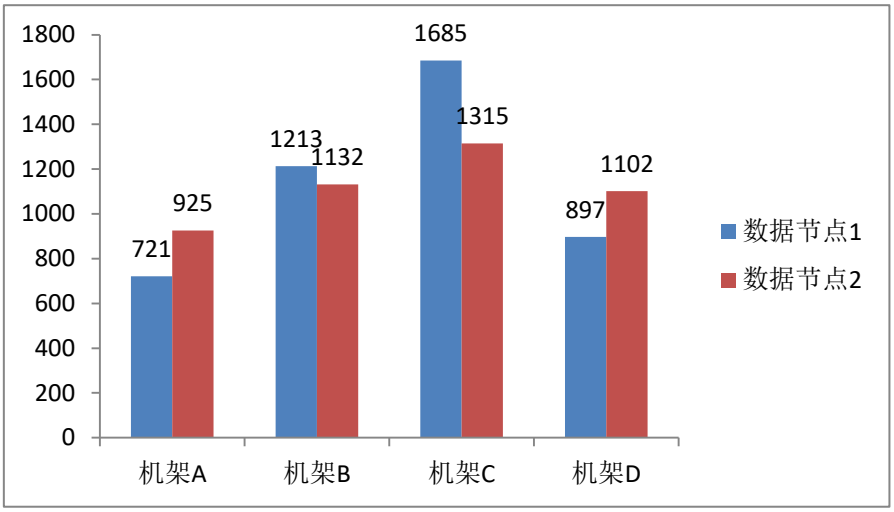

图 5.3 默认副本存放策略下的数据块节点分布

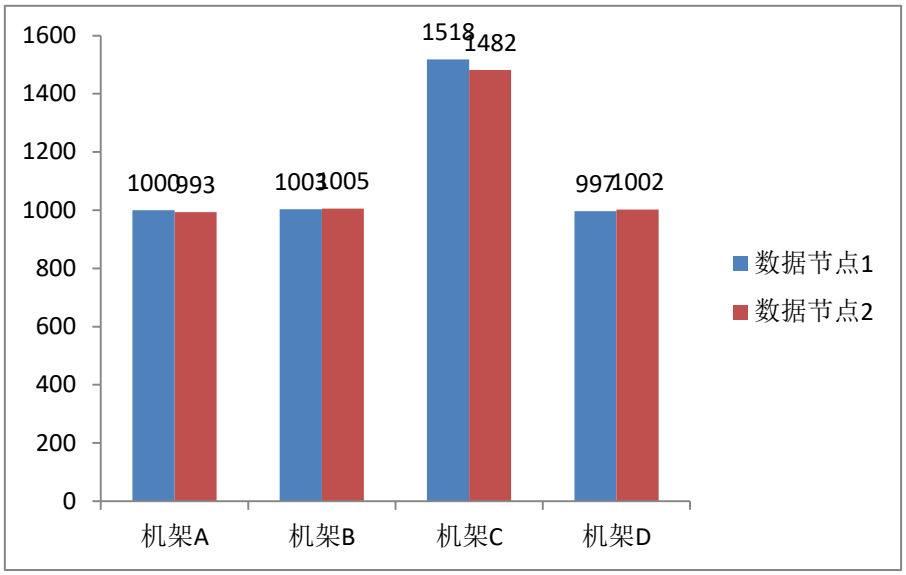

图 5.4 改进副本存放策略下的数据块节点分布

图 5.3 为默认副本存放策略下的数据块在节点中的分布情况，可以看出，任一机架中的两个 DataNode 之间都存在数据块分布不均衡的现象。对于机架 C 来说，由于采用该机架中数据节点 1 作为客户端上传数据块，根据数据块存放的本地性原则可以发现，机架 C 中数据节点 1 存放的数据块数目与数据节点 2 中存放的数目有较大差距，本地性所能够带来的好处是减少了数据块写入时需要消耗的内部带宽，带来的问题是节点负载相对较高，故这是需要进行权衡的。对于机架 A、机架 B、机架 D 来说，明显不同数据节点中数据块数目分布不均匀，这是默认副本存放策略存在的问题，也是改进副本存放策略所需要解决的问题。图 5.4 为改进副本存放策略下数据块在节点中的分布情况，很明显的是，对于机架 A、机架 B、机架 D 来说各机架中的不同节点之间负载均衡，没有出现图 5.3 中存在的问题，因此也就验证

了改进副本存放策略对于副本存放位置的选择所能够带来的影响。

当然，随着改进算法的引入，随之而来的负面影响则是选取目标节点时带来的时间上的损耗的增加。通过分析改进的算法可知，时间复杂度为 O(n),其中 n 为实验时设定的匹配节点的数目，本次实验中指定 n=5,通过实验可知，对比默认副本存放策略与改进副本存放策略针对从接收到文件存储请求直至文件存储完毕所耗时间如图 5.5 所示。

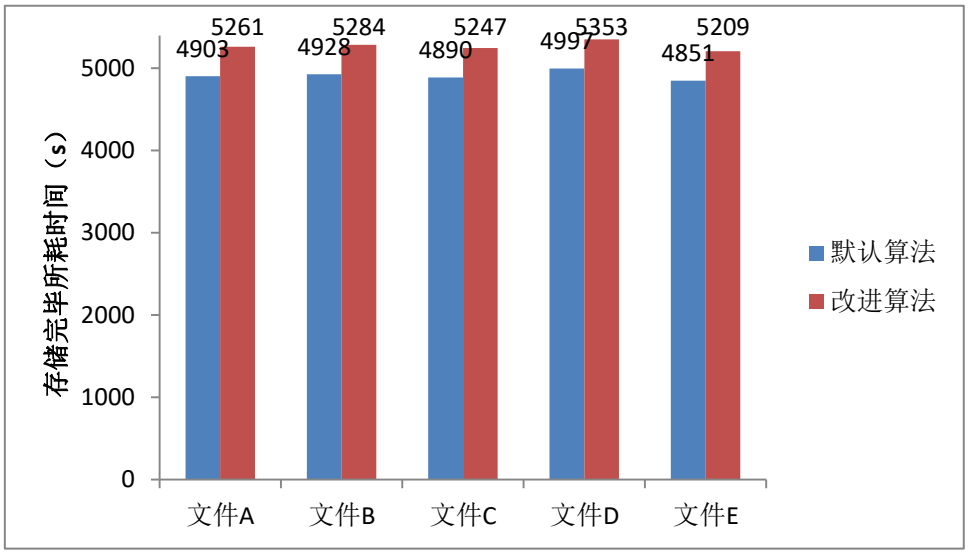

图 5.5 改进算法与默认算法所耗时间对比

通过图 5.5 发现默认算法相较与改进算法时间上存在着微弱的优势，固然节点选择会增加时间的消耗，但选择出合适的节点一方面可以实现节点负载的降低，另一方面也实现了集群内部数据块写入时内部带宽消耗的降低，因此改进算法相较于默认算法还是存在显著的优势，验证了改进算法的合理性以及正确性。

5.4 本章小结

本章描述了实验所需的软件环境以及硬件环境，通过步骤搭建 Hadoop 集群环境，验证本文研究成果的有效性。通过搭好的环境进行实验，分别对比默认副本存放策略与改进副本存放策略在节点与机架之上的情况，验证改进副本存放策略的正确性和改进副本存放策略的有效性以及所能够带来的性能提升。同时通过对比分析默认副本存放策略与改进副本存放策略文件上传时所耗时间，验证了改进的副本存放策略不会影响集群的原有性能。

第六章 总结与展望

6.1 总结

本文对 HDFS 的副本存放策略以及如何实现云存储的数据安全性展开了的研究。通过对 HDFS 的总体架构，数据流，副本管理技术的分析，指出了 HDFS 默认副本存放策略存在问题，为了在保证数据块可用性与可靠性的同时，提升数据块恢复时效率以及内部带宽的消耗，提出了一种改进的副本存放策略；针对日益高涨的数据安全性需求，为了解决用户与云存储服务商之间可能存在的发生数据泄密时的责任相互推卸，通过对可信第三方的数据安全可问责方案的引入，提出了一种基于可信第三方的 Kerberos 审计问责方案，有效地解决了云端数据发生安全性问题时的责任无法判定的问题。

本文的主要研究工作有：

（1）通过对 HDFS 系统架构的研究与分析，得出其通过将副本存放于不同机架的节点之上实现数据块的高可靠性以及高可用性，可是由于默认副本存放策略在远端机架节点选择的随机性容易造成节点之间负载不均衡以及节点失效副本恢复时带宽的巨大消耗，虽然通过 Balancer 进程可以实现将数据块从高负载的节点移至低负载节点，但并未解决数据块移动时内部带宽的消耗以及没有考虑节点异构性导致提供服务的能力的不同，因此，对默认副本存放策略的进一步改进是必要的。

（2）为了解决默认副本存放策略导致的数据恢复时内部带宽巨大消耗，本文提出改进的副本存放策略（假定副本因子数为 3），实现思路为：对于第一个副本存放节点，遵循本地节点原则，若上传文件的客户端为集群中某一 DataNode 则存放于该节点之上，否则综合考虑节点与客户端的网络距离，节点当前负载情况，节点磁盘 I/O 效率等因素来选择匹配度最高的节点作为第一个副本存放节点；对于第二个副本存放节点，以第一个副本存放节点作为参考，选取与第一个副本存放节点不同机架的节点，对于节点的匹配度判断依旧遵循上述三个要素，选取匹配列表中匹配度最高的节点作为第二个副本存放的远端机架节点；对于第三个

副本存放节点，以第二个副本存放节点为参考，选取同一机架中的不同节点作为目标节点，对目标节点计算匹配度，选取其中匹配度最高的节点作为第三个副本存放的节点。上述副本存放策略考虑了节点间负载均衡、节点服务能力以及失效副本的快速恢复。

（3）考虑到数据安全问题日益受到用户关注，本文提出一种基于可信第三方的 Kerberos 审计问责方案，以保障用户存储于云端数据的安全性以及可问责能力。本文方案实现了用户访问权限控制、用户与云端数据交互的监督以及数据安全问题发生时审计问责。通过修改 Kerberos 中 Ticket 的定义将其作为审计问责的依据；通过云端以及 KDC 存放的由用户每次请求操作生成的 Ticket 的比对来判定数据安全问题发生时的责任方；通过改进了数据传输方式，对传输的数据采用非对称加密之后再用对称加密方式，请求消息中包含时间戳字段，一旦接受消息之后发现当前时间与时间戳之差超过指定数值则认为数据传输过程中存在被窃取的可能；实现了有效的审计问责机制。

（4）通过实验对比，得出改进副本存放策略相较于默认副本存放策略的性能有较大的提升，虽然时间消耗上略有增加，但可以忽略不计，验证了改进副本存放策略的可行性。

6.2 展望

本文在优化改进的实现过程中，尚有值得进一步研究的工作。一是对于副本数目设置固定值设置方法需要进一步分类研究，如对于热点数据来说，由于访问压力的增大可能导致的问题是节点所能提供的服务能力有限，限制了服务的质量，成为性能的一个瓶颈；但对于冷门数据来说，由于副本造成了存储资源的浪费，因此动态的设定文件副本数目可能会是一个更有效的选择。二是可信第三方的资质问题会是限制该方案发展的一个至关重要的因素，如何选择一个可信的第三方，怎么判断其可信度是后续需要着重考虑的一个问题。

参考文献

[1] 孟小峰,慈祥. 大数据管理：概念、技术与挑战[J]. 计算机研究与发展. 2013,50(1):146-149.

[2] 赵黎斌. 面向云存储的分布式文件系统关键技术研究[D]. 西安:西安电子科技大学. 2011.

[3] 戴元顺. 云计算技术简述[J]. 信息通信技术. 2010(2):29-35.

[4] 陈康,郑纬明. 云计算：系统实例与研究现状[J]. 软件学报. 2009,20(5):1337-1348.

[5] Sachin Bende, Rajashree Shedge. Dealing with Small Files Problem in Hadoop Distributed File System[J]. Procedia Computer Science, 2016, Vol.79:1011-1012.

[6] 邵秀丽,王亚光,李云龙等. Hadoop 副本放置策略[J]. 智能系统学报. 2013,8(6):489-496.

[7] Feng Deng-Guo, Zhang Min, Zhang Yan, et al. Study on cloud computing security[J]. Journal of Software, 2011,22 (1):71-83.

[8] Q. Wang, C. Wang, J. Li, et al. Enabling public verifiability and data dynamics for storage security in cloud computing[C]. Proceeding of ESORICS 2009. Springer Berlin Heidelberg. 2009:355-370.

[9] Riedel E, Kallahalla M, Swaminathan R. A framework for evaluating storage system security[C]. Proceedings of the 1st Conference on File and Storage Technologies. Berkley: USENIX Association. 2002:15-30.

[10] Kamara S, Lauter K. Cryptographic cloud storage financial cryptography and data security[C]. Proceedings of the 14th International Conference on Financial Cryptography and Data Security. Berlin: Springer. 2010:136-149.

[11] 李建华. 现代密码技术[M]. 机械工业出版社. 2007

[12] 陈晨,陈达丽. 谷歌大数据技术的研究及开源实现[J]. 软件产业与工程. 2015,1(5):31-36.

[13] 胡文波,徐造林. 分布式存储方案的设计与研究[J]. 计算机技术与发展. 2010(04):65-68

[14] 文莎. 分布式文件系统综述[J]. 软件导刊. 2015(11):27-29

[15] 郭耀华. 基于云存储的数据存储系统的设计与实现[D]. 北京:北京邮电大学. 2012.

[16] 崔培枝,王朝君,刘海燕. Kerberos 认证技术研究及分析[J]. 计算机与现代化. 2001(5):35-39.

[17] 陈贞. HDF 环境下的访问控制技术研究[D]. 重庆:重庆大学. 2013.

[18] 许信. 云存储系统服务质量控制与可靠性技术研究[D]. 杭州:浙江大学. 2011.

[19] 赵黎斌. 面向云存储的分布式文件系统关键技术研究[D]. 西安:西安电子科技大学. 2011.

[20] 王玲惠,李小勇,张轶彬. 海量小文件存储文件系统研究综述[J]. 计算机应用与软件. 2012, 29(8):106-109.

[21] 翟猛. 基于 TFS 的分布式文件存储平台研究与实现[D]. 天津:河北工业大学. 2014.

[22] 王子伟,王晓京. 基于低密度随机纠删码的 TFS 容灾优化方案[J]. 计算机应用. 2016,36(S2):66-68.

[23] 童明. 基于 HDFS 的分布式存储研究与应用[D]. 武汉:华中科技大学. 2012.

[24] Zhendong Cheng, Zhongzhi Luan, Alain Roy, et al. ERMS: An Elastic Replication Management System for Hadoop[C]. Proceedings of IEEE International Conference on Cluster Computing Workshops. 2012:32-40.

[25] K. Sashi, Antony Selvadoss Thanamani. A New Replica Creation and Placement Algorithm for Data Grid Environment[C]. Proceedings of IEEE International Conference on Data Storage and Data Engineering. 2010:265-269.

[26] Krish K.R, Aleksandr Khasymski, Ali R Butt, et al. AptStore: Dynamic Storage Management for Hadoop[C]. Proceedings of IEEE International Conference on Cloud Computing Technology and Science. 2013:33-41.

[27] Andreas Haeberlen. A case for accountable cloud[J]. ACM SI-UOPS Operating Systems Review. 2010, 44(2):52-57.

[28] Siani Pearson. Toward accountability in the cloud[J]. IEEE Internet Computing. 2011,15(4):64-69.

[29] Ryan K L Ko, Bu Sung Lee1, Siani Pearson. Towards achieving accountability, auditability and trust in cloud computing[M]. Advances in Computing and Communications, Springer Berlin Heidel-berg. 2011:432-444.

[30] Lotz V, de Oliveira A S, Sendor J. Control as a means towards accountable services in the cloud[J]. Computer systems science and engineering. 2013,28(6):377-386.

[31] Zou J, Wang Y, Orgun M A. Modeling Accountable Cloud Services Based on Dynamic Logic for Accountability[J]. International

Journal of Web Services Research. 2015,12(3):48-77.
[32] 王永洲. 基于 HDFS 存储技术的研究[D]. 南京:南京邮电大学. 2013.
[33] 曹卉. Hadoop 分布式文件系统原理[J]. 软件导刊. 2016,15(3):15-17.
[34] Wenzhe Liao. Application of Hadoop in the Document Storage Management System for Telecommunication Enterprise[J]. International Journal of Interdisciplinary Telecommunications and Networking (IJITN). 2016,8 (2):58-68.
[35] 张波. HDFS 下文件存储研究与优化[D]. 广州:广东工业大学. 2013.
[36] 邹振宇,郑烇,王嵩,杨坚. 基于 HDFS 的云存储系统小文件优化方案[J]. 计算机工程. 2016(03):31-40+46.
[37] 李宽. 基于 HDFS 的分布式 Namenode 节点模型的研究[D]. 广州:华南理工大学. 2011。
[38] 黄晓云. 基于 HDFS 的云存储服务系统研究[D]. 大连:大连海事大学. 2010。
[39] 王意洁, 许方亮, 裴晓强. 分布式存储中的纠删码容错技术研究[J]. 计算机学报. 2017(1):236-255.
[40] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, etal. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility[J]. Future Generation Computer Systems. 2009,25(6):599-616.
[41] 孙健,贾晓菁. Google 云计算平台的技术架构及对其成本的影响研究[J]. 电信科学. 2010,26(1):38-44.
[42] 杨帆. Hadoop 平台高可用性方案的设计与实现[D]. 北京:北京邮电大学. 2012.
[43] 王鲁俊,龙翔,吴兴博,王雷. SFFS：低延迟的面向小文件的分布式文件系统[J]. 计算机科学与探索. 2014,8(4):438-445.
[44] 段效琛,李英娜,贾会玲,赵振刚,李川. 初始信息素筛选的蚁群优化算法在 HDFS 副本选择中的研究[J]. 传感器与微系统. 2017,45(04):31-33.
[45] 王来,翟健宏. 基于 HDFS 的分布式存储策略分析[J]. 智能计算机与应用. 2016,6(1):6-8.
[46] 李晓恺,代翔,李文杰等. 基于纠删码和动态副本策略的 HDFS 改进系统[J]. 计算机应用. 2012,38(8):2150-2153+2158.

附录 1 攻读硕士学位期间撰写的论文

[1] 周长俊,宗平. Hadoop 副本存放策略的改进. 计算机技术与发展. 已录用.

致谢

不知不觉，两年半的研究生生活即将过去，回首这段学习经历，感到特别的充实。我的导师宗平教授亲切的指导和照顾、师门兄弟姐妹们的无微不至关怀，让我的研究生生活收获知识的同时还有友情。

首先我要感谢宗老师在两年半时间里对我的指导和关怀，宗老师为人特别和蔼可亲。从研一开学，宗老师认真地给我们讲解了学科特点，并指导我们如何选课。在论文选题的时候，宗老师极其认真地为我们选择的课题，让我少走很多弯路。论文的写作过程中，宗老师一直陪伴着我一起走过，每当我遇到问题，宗老师总是不厌其烦的为我讲解。宗老师教导我的不止于书面的的知识，为人处事方面同样让我收获颇丰。值此之际，请允许我向宗老师表达最真挚的感谢和最诚挚的敬意。

感谢教研室与我一起共同学习、生活的陈洁、蒋君妍以及梁胜昔，感谢他们在学习和生活中对我的帮助，感谢他们的一路陪伴。

感谢室友薛拾军、刘中锋以及候学东，感谢他们给与我的帮助，是你们让我感受到了家的温暖，在我遇到困难时，给我最真诚的建议。

最后，感恩父母，感谢你们长久以来对我的关心和照料。当我遇到困难的时候总是站在我面前为我遮风挡雨，为了我，你们付出了全部的心血，感谢你们对我的无私付出。

感谢参考文献的相关学者，他们的研究思路和学术见解让我获益匪浅。

感谢百忙之中对我的论文进行评阅、提出修改意见的老师和专家们。