**密级： 保密期限：**

xm 拷贝

**硕士学位论文**



**题目：适用于列式数据库的大数据存储机制研究**

**学 号：**  **2015110002**

**姓 名： 李开放**

**专 业： 信息与通信工程**

**导 师： 别红霞**

**学 院： 信息与通信工程**

**2017年12月1日**

独创性（或创新性）声明

本人声明所呈交的论文是本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中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他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。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

保密论文注释：本学位论文属于保密在年解密后适用本授权书。

非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

导师签名：日期：\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

适用于列式数据库的大数据存储机制研究

**摘 要**

近些年来，随着互联网技术的飞速发展，数据量呈爆炸性趋势增长，不断地向大数据存储技术发起挑战。数据存储作为大数据行业的重要环节之一，一直是业内学者们研究的重点。分布式存储将海量数据分散存储在多台服务器节点上，相比传统的集中式数据库，不仅大大提高了存储容量和数据处理能力，还具有更高的容错性和可靠性。

目前，HDFS是最受用户欢迎的分布式存储系统之一。该存储系统在整体上基本保证了各节点之间存储数据量的均衡，同时具有较高的存储可靠性。不过，HDFS对于列式数据而言，并不能保证每个字段在各节点之间的均衡，且其副本存储方案在可靠性方面还存在一定的改善空间。本文主要针对这两方面问题进行研究，提出了一种新的分布式存储算法，并进行优化实现。本文主要工作如下：

（1）针对HDFS分布式存储算法不能保证列式数据各字段在节点之间均衡存储的问题，提出了一种基于字段均衡的分布式存储调度策略，使得列式数据库中每个字段在各节点之间的存储最多相差一个数据块的均衡效果。

（2）在分布式数据存储过程中，使同一数据块的不同副本跨交换机存储，提高了数据存储的可靠性。

（3）针对集群新增节点的情况，提出一种负载均衡算法，在保证副本跨交换机存储的原则上，使新增节点之后数据存储重新达到基于字段均衡的状态，并通过实验证明了该算法能够使得每个字段在各节点之间的存储相差不超过1个数据块。

**关键词：**数据存储 字段均衡 可靠性 负载均衡

RESEARCH ON LARGE DATA STORAGE

MECHANISM FOR COLUMN DATABASE

**ABSTRACT**

In recent years, with the rapid development of Internet technology, the amount of data presents explosive growth, and constantly challenges the big data storage technology. As an important part of the big data industry, data storage always is the focus of industry scholars. Distributed storage can store large amounts of data on multiple server nodes. Compared with traditional centralized database, distributed storage not only improves storage capacity and data processing ability, but also has higher fault tolerance and reliability.

At present, HDFS is one of the most popular distributed storage systems. The storage system basically guarantees the balance of data storage between nodes, and has higher storage reliability. However, HDFS does not guarantee the equilibrium of each field among the nodes for the column data, and the replica storage scheme does not achieve the highest storage reliability. This thesis focuses on these two aspects of research, and puts forward a new distributed storage algorithm, and optimize the implementation. The main work of this thesis is as follows:

（1）According to the HDFS distributed storage algorithm cannot guarantee the column data of each field storage balance between nodes, this thesis proposes a distributed storage algorithm based on balanced field, every field data for each column storage database at each node between the maximum difference of a data block equalization effect.

（2）In the process of distributed data storage, the different copies of the same data block are stored across switches, which improves the reliability of data storage.

（3）According to the new cluster node, this thesis proposed a load balancing algorithm in the principle of ensuring replica cross switch storage, to reach the clusters a balanced state based on the field, and the experiments proved that the algorithm can make each field in each node stored between the difference of not more than 1 block of data.

**KEY WORDS:** data storage field equalization reliability load balancing

**目 录**

[第一章 绪论 1](#_Toc499802867)

[1.1 课题背景 1](#_Toc499802868)

[1.2 课题研究现状 2](#_Toc499802869)

[1.2.1列式数据库的发展 2](#_Toc499802870)

[1.2.2分布式存储系统的发展与现状 3](#_Toc499802871)

[1.3 课题研究内容 5](#_Toc499802872)

[1.4 文章组织结构 6](#_Toc499802873)

[第二章 相关研究工作 7](#_Toc499802874)

[2.1分布式存储系统 7](#_Toc499802875)

[2.1.1分布式存储系统结构 7](#_Toc499802876)

[2.1.2分布式存储系统的工作原理 7](#_Toc499802877)

[2.1.3分布式存储系统特性 8](#_Toc499802878)

[2.1.4分布式存储系统的关键技术 9](#_Toc499802879)

[2.2相关技术和工具 10](#_Toc499802880)

[2.1.1 文件系统设计 10](#_Toc499802881)

[2.2.2 Nanomsg传输协议 11](#_Toc499802882)

[2.2.3数据备份策略 12](#_Toc499802883)

[2.2.4 HDFS中存储调度算法 12](#_Toc499802884)

[2.3负载均衡技术 13](#_Toc499802885)

[2.3.1负载均衡的类别和意义 13](#_Toc499802886)

[2.3.2负载均衡算法 14](#_Toc499802887)

[第三章 基于字段均衡的分布式存储调度算法 16](#_Toc499802888)

[3.1基于字段均衡的分布式存储算法 16](#_Toc499802889)

[3.1.1HDFS中分布式存储调度研究 16](#_Toc499802890)

[3.1.2基于字段均衡的分布式存储算法 18](#_Toc499802891)

[3.2字段均衡性 20](#_Toc499802892)

[3.3存储可靠性 21](#_Toc499802893)

[3.3.1集群稳定性 21](#_Toc499802894)

[3.3.2数据故障处理 22](#_Toc499802895)

[3.4数据存储与访问的效率 28](#_Toc499802896)

[3.5存储堆积问题及修正 29](#_Toc499802897)

[3.5.1部分字段存储堆积问题 29](#_Toc499802898)

[3.5.2算法优化 29](#_Toc499802899)

[1.5 3.6小结 31](#_Toc499802900)

[第四章 新增节点后的负载均衡策略 33](#_Toc499802901)

[4.1算法步骤 33](#_Toc499802902)

[4.2负载均衡分类 34](#_Toc499802903)

[4.2.1新增节点部署在原交换机下 34](#_Toc499802904)

[4.2.2新增节点部署在新交换机下 35](#_Toc499802905)

[4.3小结 36](#_Toc499802906)

[第五章 仿真结果与分析 37](#_Toc499802907)

[5.1字段均衡的性能与分析 37](#_Toc499802908)

[5.1.1字段均衡性 37](#_Toc499802909)

[5.1.2大量不同字段存储优化 41](#_Toc499802910)

[5.2新增节点后负载均衡仿真 41](#_Toc499802911)

[5.3小结 43](#_Toc499802912)

[第六章 结论与展望 44](#_Toc499802913)

[6.1结论 44](#_Toc499802914)

[6.2不足与展望 45](#_Toc499802915)

[参考文献 46](#_Toc499802916)

[致谢 50](#_Toc499802917)

[攻读学位期间取得的研究成果 51](#_Toc499802918)

第一章 绪论

* 1. 课题背景

数据是信息化时代的基本要素之一，计算机处理和应用的一切对象都与数据有着密不可分的关系。对于企业而言，海量数据中蕴藏着许多重要的信息，企业保存数据就是为了从中挖掘出更多有价值的信息。例如，零售商可以从交易记录中找到客户购买习惯和购买模式的重要信息；医院可以从病人的数据中提取患者的相关信息来帮助治疗；银行可以通过客户的交易记录中或者金额交易信息，保障其经济安全。正因为数据蕴含着这么多有价值的信息，对用户非常重要，所以其存储和处理技术也越来越被业界重视[1]。

存储系统的快速发展可以追溯到上个世纪70年代。1978年，IBM首先实现了分级存储管理（Hierarchical Storage Management，HSM）的大型机系统[2]。HSM是一种将离线存储与在线存储结合起来的技术，它将磁盘中常用的数据按指定的策略自动迁移到磁带库等二级大容量的存储设备中。当用户请求这些数据时，分级存储系统会自动将这些数据从下一级的存储设备调回到上一级存储设备上。由此开始，存储系统经历了迅猛的发展。

近些年来，虽然存储技术的发展日新月异，却难以满足数据量的爆炸性增长带来的更高的要求。存储技术在发展的同时，数据量却以更快的速度增长。IDC最新调研报告指出，从现在开始到2020年，数字宇宙的膨胀率大概为每两年翻一番。据此粗略估计，到2020年，“数字宇宙”将会膨胀到40000EB，约40万亿GB[3]。如果使用2T的硬盘进行存储，则需要200亿个。对于如此海量的数据，传统的集中式存储系统越来越显力不从心，不仅难以在存储空间、散热等物理因素上满足需求，尤其是在处理高并发的用户请求、保障数据安全可靠等性能标准上难以支持。针对集中式存储系统所暴露出的缺点，越来越多的用户开始把目光转移到分布式存储系统上[4]。

分布式存储采用了可伸缩扩展的系统架构，使用多台独立的小型存储设备共同分担数据存储任务。它与传统的集中式存储恰恰相反，将物理存储介质部署在不同的位置，这些小型的存储设备可以分散在企业的各个角落，只是在整体上构成一个虚拟的存储系统。理论上来说，在一定范围内，只要增加节点的数量，其存储容量还是非常可观的。但是，数据存储并不仅是将数据存储在服务器中，还要保证其安全、可靠等性能。虽然分布式存储已经在很大程度上改善了数据存储洗可靠性，但是目前在各个行业中，数据损坏的问题仍然不容小觑。图1-1展示了2015年部分行业平均每小时由于存储系统故障而造成的经济损失[5]。由图中数据可知，数据存储系统故障影响到诸多行业，带来的损失也非常大，同时说明了数据可靠存储的重要性。



图1-1 存储系统故障为各行业带来的损失

在存储容量上，虽然分布式文件系统比传统的集中式数据库增大很多，但是随着数据不断产生，文件系统常有容量不足的时候。此时，在不删除原有数据的基础上，若在同一个数据库中继续存储数据，就必须对集群进行扩容。在过去的十年里，人们在开始使用特定策略比如服务器整合、增加磁盘容量和主机核数等方法，纵向扩展数据库规模[6]。然而，纵向扩展架构几乎无法提供存储资源的线性增长，例如进行纵向扩展的通用虚拟环境使用了尽可能少的物理服务器，在处理器远没有到达极限之前，内存以及磁盘容量已经全部耗尽了，并没有实现处理器资源的价值最大化[7]。随着工作负载不断增长以及新需求的出现，横向扩展架构正在重新出现并影响着分布式数据库的发展。横向扩容指在原来集群基础上，增加集群节点数量，所以在一定的范围内，横向扩容对集群各方面的影响几乎是线性增长的，比如存储容量，解决了用户对存储最迫切的需求之一。但是集群横向扩容打破了原来数据均衡存储的状态，所以为了使数据存储重新基于节点均衡，新增节点后的负载均衡也成为了集群管理重要的一部分[8]。

* 1. 课题研究现状

1.2.1列式数据库的发展

早在十年前的互联网企业中，数据库的规模就达到了100GB左右[9]，为了满足更多数据的存储需求，部分企业率先开始挖向TB级数据管理及分析发起了挑战。列式数据库在沉默了将近十年之后，重新被一些眼光前瞻的用户关注。

有关列式数据库的发展，可以追溯至1994年Sybase公司收购了一家名为Express Way Technologies的公司。该公司主要生产帮助传统数据库做报表加速的产品，即把行式数据库的数据转换为列的形式进行存储。后来，基于该技术的Sybase IQ产品在1996年被正式推出，并一直延续至今[10]。

与列式数据库对应的是行式数据库。顾名思义，行式数据库以“行”为基准把一行的各个字段数据连续存储，这样当用户对以行为单位的数据进行操作时确实直观高效[11]。但是，随着用户需求越来越多样化，行式数据库的弊端也暴露出来。当用户对数据库中的数据进行统计分析时，用到的往往只是某些字段的数据，并不需要把每行的所有字段都读取过来。比如，某行式数据库中存储了某城市一年的空气监测数据，其中包括温度、湿度、PM2.5、风速、时间、位置等指标，当用户研究分析该城市一年内PM2.5变化情况时，行式数据库会把其他字段的数据也读取到内存，浪费了宝贵的I/O资源。随着类似需求越来越多，列式数据库应运而生。

列式数据的存储方式就是把行式数据库中每一行各个字段的数据拆分开来，以列的形式重新组合进行存储，即将所有行的某字段重新组合为一列。所以当对数据库中某些字段进行分析时，只读取相应的列即可[12]。仍然以分析某城市PM2.5变化情况为例，在列式数据库中无需温度、湿度、风速等无关字段，降低了磁盘读取压力，提高了数据分析效率。另外，在列式数据库中每列数据为同一字段，尤其像时间、地点等这类字段重复率很高，更容易压缩[13]。所以，列式数据库相比行式数据库而言，还能够节省更多的存储空间。

随着互联网的发展，如今的列式数据库在数据集市、企业商务智能（BI）和数据仓库等领域发挥着越来越重要的作用，尤其是在全球各大企业中支撑着海量的大数据分析场景的应用。现在几乎所有的数据库厂商都接受了列数数据存储技术，并在此基础上研究推出不同类型和风格的列式存储产品，刚投入应用不久的微软SQL Server 2012中的Columnstore Index，就是一个典型的例子[14]。

列式数据库的出现和发展不仅有效地降低了系统I/O，还充分发挥了字段内部的压缩优势，为数据存储节省了大量的空间。同时，这也是今天列式数据库受广大用户欢迎的重要原因。

1.2.2分布式存储系统的发展与现状

随着分布式存储系统的发展，越来越多的企业公司开发出了适用于自己的分布式存储系统。目前，比较流行的分布式存储系统主要有谷歌的谷歌文件系统（Google File System，GFS）[16]、微软的Azure[17]、亚马逊的Dynamo[18]和Apache的HDFS[19]。其中HDFS是GFS的开源实现，作为后台的基础设施广泛应用于众多大型企业，如Yahoo、Amazon、Facebook、eBay等[20]。

分布式存储系统在20世纪70年代开始崭露头角，然后逐渐延伸至各个领域。从最初的网络文件系统（Network File System，NFS）到现在的存储池[21]，分布式存储技术在规模、结构、性能等方面发生了巨大的变化。

第一代分布存储系统主要以提供标准结构的远程文档访问为目的，比较重视访问的性能和数据存储可靠性。其中比较具有代表性的有NFS和安德鲁文件系统（Andrew File System，AFS）等，而且它们对后来的分布式系统有很大的影响[22]。从1985年至今，NFS主要经历了四次重要的版本更新，几乎可以移植到任何主流的操作系统中。NFS利用Unix系统中的虚拟文档系统（Virtual File System，VFS）机制，通过远程过程调用和规范的文档访问协议将客户机的请求转发到服务器进行处理，服务器在VFS上通过本地文档系统对文档处理，从而实现了全局的分布式文档系统[23]。后来，Sun公司公布了NFS的实施规范，互联网工程任务组（The Internet Engineering Task Force，IETF）也将其列为征稿意见，这些在很大程度上都促进了NFS的流行[24]。AFS在1983年诞生于美国卡耐基梅隆大学，该系统将分布式文档系统的扩展性放在了设计和实现的首要位置，并完善了在不安全的网络中的安全访问技术[25]。另外，AFS拥有较强的扩展性，比较适合数百个甚至上千个节点的分布式环境。在大规模的分布式系统中，AFS能够利用本地存储作为分布式文档的缓存，即便在远程文档不可访问的情况下，AFS仍然能够部分工作，大大提高了系统的可用性。在可用性和安全性方面，AFS对后来的Coda File System、Inter-mezzo File System都有很大的影响，尤其是Coda，在支持移动计算方面做了很多研究工作。早期的分布式存储系统主要以提供结构的远程文档访问为重点，在受磁盘、网络、处理器等各方面条件限制的情况下，更多地关注数据存储的可靠性和数据访问的性能。NFS和AFS在系统结构方面进行了很多探索，它们采用的很多技术也都被后来的分布式存储系统所借鉴。

20世纪90年代初，人们迎来了第二代分布式存储系统。其中比较具有代表性的有XFS、Tiger Shark并行文档系统、基于虚拟共享磁盘Petal的Frangipani分布式文档系统和分层文件系统（Slice File System，SFS）[26]。

面对广域网和大容量存储应用的需求，加利福尼亚大学的Silicon Graphics基于当时的高性能对称多处理器的设计思想，设计研发了XFS[27]。XFS克服了以往的分布式存储系统中一般都运行在局域网上的弱点，解决了在广域网上的缓存问题。XFS中所采用的多层次结构很好地利用率文档系统的局部访问特性，很大程度上减轻了网络负载，提高了对本地主机和本地存储空间的有效利用率，所以拥有比较出色的性能。

Tiger Shark并行文档系统主要是针对大规模实时多媒体应用设计的，其中使用了大量技术确保多媒体传输的稳定性和实时性[28]。

基于虚拟共享磁盘Petal的Frangipani分布式文档系统率先采用了分层次系统的服务，使得存储系统在设计上得到了简化。在Frangipani中，每个客户端也被设计为文档系统服务器参与文档管理，并通过分布式锁机制实现同步访问控制[29]。分层结构的分布式存储系统具备更好的扩展性，且能够在线动态地增添存储设备、用户、备份等，在处理节点失效、网络故障等方面也具备较完善的机制，所以其系统可靠性比较可观。

SFS主要特点是它考虑了标准的NFS在存储容量和性能方面的限制，在客户机和服务器之间架设了一个μproxy中间转发器，从而提高了系统的性能和扩展性。该系统将客户端的请求分为三种：小文档、元数据服务和大文档数据。μproxy中间转发器将小文档和元数据服务请求转发到不同的文档服务器上，将大文档数据请求发送至存储服务器上。这样，SFS可以同时支持多个存储服务器，提高了整个系统的容量和性能[30]。

20世纪90年代的后5年，分布式存储系统跨入了第三个发展阶段。由于网络技术的飞速发展，极大地推动了分布式文档系统的研究，存储技术也有了很大的进步。在这个阶段，虽然单位存储的成本大幅度降低，但是数据总线带宽、磁盘速度的增长却无法满足要求，所以存储子系统成了分布式存储系统的瓶颈。这一时期，出现了多种分布式存储系统，如RedHat公司的Global File System、General Parallel File System（GPFS）、惠普的DiFFS等[31]，其中应用了比较先进的分布式锁、缓存管理、数据负载平衡等技术。这一时期的分布式存储系统规模更大、系统更复杂，对物理设备的直接访问、磁盘布局和检索效率的优化、元数据的集中管理等都反映了对性能和容量等需求的满足。

2000年以后，随着SAN和NAS逐渐成熟，人们开始考虑如何将两种结构结合起来，充分利用两者的优势。另外，传输协议的探索也是研究的热点之一，Direct Access File System利用了远程内存直接访问的特性，借鉴了Common Internet File System等协议，设计出了一份新的网络文件访问协议[32]。这一时期，比较具有代表性的分布式存储系统有IBM的Storage Tank、Cluster的Lustre、Panasas的PanFS、蓝鲸文件系统（Blue Whale File System，BWFS）等[33]。这些存储系统的体系结构趋于一致，设计策略也大概相同，相比之前的存储系统，都取得了更为出色的性能。

* 1. 课题研究内容

本课题主要对适用于列数据库的分布式存储调度算法进行研究分析，针对数据在集群节点上的存储均衡问题，提出了基于字段均衡的分布式存储算法和新增节点之后的负载均衡方法，并对该存储策略和HDFS中分布式存储调度策略进行仿真对比分析。另外，该课题研究分析了分布式存储系统中集群扩容时的负载均衡策略以及集群的灾难恢复，并进行仿真实现。

论文主要工作从以下几点展开：

1. 研究HDFS中分布式存储调度算法；
2. 提出基于字段均衡的分布式存储调度算法；
3. 集群故障时存储系统灾难恢复研究；
4. 集群新增节点后负载均衡的研究与优化；
5. 基于字段均衡的分布式存储、负载均衡等算法的验证结果与分析。
6. 论述课题研究的不足之处，并提出改进方案。
   1. 文章组织结构

根据论文的工作与研究内容，本文的结构按照如下进行安排：

第一章：绪论。主要介绍分布式文件存储系统的研究背景和发展现状，阐述本文课题的主要研究内容，并列出论文内容的组织结构。

第二章：相关研究工作。介绍了分布式存储系统的基本架构和关键技术，以及课题研究工作中所使用的相关技术与工具。

第三章：基于字段均衡的分布式存储调度算法。一般的分布式存储策略在存储时只保证了节点上存储容量的基本均衡，然而对于列式数据库而言，一般的任务调度都是针对字段而言的，所以本章提出了一种基于字段均衡的分布式存储策略，保证数据的各个字段在每台节点上基本存储均衡。另外，本章还对大量不同字段存储时可能引起的数据堆积问题进行了研究，并提出了解决方案。

第四章：集群新增节点时的负载均衡策略。随着存储数据量的增长，集群有时通过增加节点来扩大数据库容量。但是集群新增节点打破了原来数据的存储均衡状态，况且为了尽可能地利用新增的节点资源，需要将原节点中的部分数据迁移到新增节点上。本章提出的负载均衡策略所实现的功能就是使集群在新增节点之后重新调整到各节点之间基于字段均衡。

第五章： 仿真结果与分析。本章首先对基于字段均衡的分布式存储算法及其优化后的方法进行实现，并对比HDFS中的分布式存储算法仿真实现结果，验证了利用该算法在数据存储完成之后的均衡性能。另外，本章还仿真实现了集群新增节点时候的负载均衡，验证了该算法的可行性与均衡性能。

第六章：总结与展望。对本课题的工作成果进行总结，并指出其中不足之处的改进优化方案。

第二章 相关研究工作

2.1分布式存储系统

2.1.1分布式存储系统结构

分布式存储系统主要由分布在各个位置的独立节点和交换机等设备组成，通过有线连接节点之间能够进行通信。如图1-2，展示了一般的分布式存储系统的整体架构。分布式存储系统主要解决了单机存储所存在的容量受限、数据处理能力较低的问题。通过把数据分散存储在多台设备上，分布式系统拥有大容量、高性能、高可用、扩展性好的存储服务[34]。



图1-2 分布式存储系统架构

2.1.2分布式存储系统的工作原理

从角色上讲，分布式存储系统一般由一台Master节点、多台Chunk服务器组成。系统可以同时被多台客户端访问，处理高并发的用户请求[35]。如图1-3所示，以GFS为例展示了分布式存储系统中各角色的工作内容。

在GFS中，大文件被分割为固定大小的块（Chunk）。Chunk服务器把分割后的Chunk存储在本地磁盘上。为了保证数据存储可靠性，GFS默认把每份Chunk复制3份，用户也可以指定备份数目。

在集群中，Master节点像大脑一样控制着整个系统的运作。GFS中所有文件系统原数据由Master节点管理，比如命名空间、文件和Chunk的映射信息、Chunk位置信息等。另外，Master节点还管理着系统中块的租用管理、孤立块的回收以及负载均衡时Chunk在不同服务器之间的移动。Master节点与每个Chunk服务器通过心跳机制进行通讯，保证了集群的正常运行[36]。

图1-3同时也展示GFS中在单点Master下一次简单的数据读取过程：

（1）客户端向Master节点发送请求信息，其中包含文件名、Chunk索引等信息。

（2）Master节点借助心跳检测模块监测Chunk服务器，并向Chunk服务器发送指令。

（3）Chunk服务器周期性地将自己的状态发送到Master节点。

块服务器需要周期性的返回自己的状态给Master，以确保能够接收Master的请求。

（4）Master节点将Chunk信息（Chunk句柄、Chunk位置等）发送给客户端。

（5）客户端根据Chunk信息，无需再通过Master节点，直接在Chunk服务器中查找最近的Chunk副本。

（6）Chunk服务器将相应的Chunk数据发送给客户端。

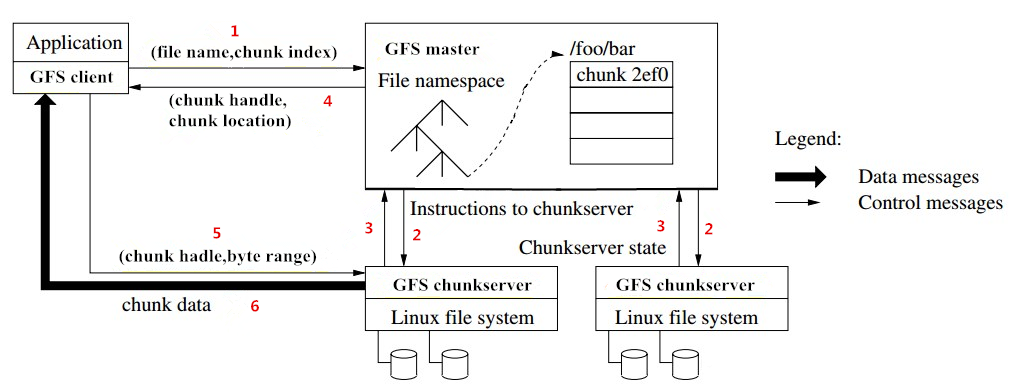


图1-3 GFS基本架构示意图

2.1.3分布式存储系统特性

不同的分布式存储系统相互之间都存在一定的差别，但是，标准的分布式系统在没有特定业务逻辑约束的情况下，一般都会具备以下特征：

（1）分布性

不同于集中式存储系统，分布式系统中的存储设备可以在空间上随意分布，方便了系统在物理空间上搭建设计。

（2）扩展性

分布式存储系统中的多台Chunk节点数目可以随着存储容量的需要而增加，这也是该系统能够解决海量数据存储问题的关键之处。

（3）对等性

分布式存储系统中的Chunk副本没有主从之分，副本是系统为保障数据可靠性的一种形式，副本中信息都是一致的，所以当某节点上的数据丢失时，系统可以从其他副本获取相应的数据信息，这是解决分布式系统中数据故障最有效的手段[16]。

（4）并发性

在计算机网络中，程序的并发性操作是很常见的行为。由于分布式系统需要支持多个客户端同时访问，可能会并发性地操作一些共享资源，所以如何准确高效地协调分布式存系统中的并发性操作成了分布式系统中的一大挑战。

（5）故障常态

在整个分布式数据存储系统中，涉及了大量设备，无论磁盘还是交换机，都有一定的几率发生故障。随着系统规模的增大，系统中设备故障就成了很“正常”的情况，所以分布式存储系统必须能够应对这种故障，才能保证数据的可靠性，满足用户的存储需求[16]。

2.1.4分布式存储系统的关键技术

在分布式文件系统中，涉及了很多前沿而且新颖的技术，其中比较彰显分布式系统特色的主要有：分布式文件系统、分布式计算、分布式数据库。

（1）分布式文件系统

分布式文件系统在整个存储体系中处于最底层。从用户角度来看，分布式文件系统是一个提供了一系列API的标准文件系统，保证了文件的创建、删除、移动等基本操作。从实现原理来看，分布式文件系统不像普通的文件系统管理本地磁盘，因为其文件内容和目录结构可能分布式存储在远端节点上。这种将文件分布式存储在一簇机器上的特点是分布式存储系统与一般存储系统关键的区别之一。

（2）分布式计算

关于分布式计算，最常见的例子就是MapReduce。分布式文件系统就是为分布式计算服务的。一般情况下，当输入文件过大时，单机计算效率低下甚至完全不可支持，分布式计算将任务扩展到大规模的集群上进行，将待处理的数据集分解为小数据集，然后将这些小数据集并行处理，从而提高了数据库工作效率[23]。

（3）分布式数据库

与传统数据库相比，分布式数据库主要具备两个特点：可扩展性和高可靠性。传统数据库由于其高一致性导致它的扩展性非常复杂，而分布式数据库弱化了数据处理过程中的一致性，保证了最终数据的一致性，这样既扩展了数据库规模，又满足了企业的应用需求。

2.2相关技术和工具

2.1.1 文件系统设计

不管是传统的集中式数据库还是现在的分布式文件存储系统，元数据管理都是数据库中数据存储的核心工作。在海量数据的存储中，元数据的量也很大，因此对元数据文件系统的设计是实现一个数据库引擎的重要环节。

在分布式文件存储系统中，文件都是被切分为固定大小的数据块（DataBlock，DB）存储在各个节点上的，所以集群必须妥善管理服务器控制信息、节点信息、字段信息、DB存储信息等。为了管理集群中的相应数据文件，课题中进行了以下数据结构设计：

（1）DB位置信息：

当用户上传数据时，首先向集群发起请求，集群经过调度计算得出DB存储位置DataBlockLocation，即ID为NodeId的节点上DB索引为DBIndex的数据块第OffSet条位置。

|  |  |  |
| --- | --- | --- |
| 变量名 | 含义 | 类型 |
| NodeId | 节点ID | short |
| DBIndex | DB索引 | int |
| OffSet | DB偏移量 | unsigned long |

（2）节点信息：

对于集群中节点信息的管理，在程序设计时主要维护以下信息。

|  |  |  |
| --- | --- | --- |
| 变量名 | 含义 | 类型 |
| NodeId | 节点ID | short |
| Pi | 节点性能参数 | float |
| MacAdderss | 服务器节点Mac地址 | char[] |
| SwitchNum | 所在交换机序号 | short |
| DBStoredNum | 已存储DB数目 | int |

（3）节点上的DB存储信息：

DB在节点上存储完成之后的相关存储信息都会被集群管理节点记录下来，系统即通过该表对各个DB的访问。

|  |  |  |
| --- | --- | --- |
| 变量名 | 含义 | 类型 |
| NodeId | 节点ID | short |
| Adderss | DataBlock起始地址 | unsigned long |
| CurrentOffSet | 该DB当前存储记录条数 | long |
| DataBaseId | 数据库ID | short |
| TableId | 表ID | short |
| FieldId | 字段ID | int |
| CopyNum | 副本ID | short |

（4）分布式文件系统索引表

分布式文件系统索引表（Distributed File System Index Table，DFS\_IT）管理整个集群中的字段信息，主要记录每个字段的存储容量。

|  |  |  |
| --- | --- | --- |
| 变量名 | 含义 | 类型 |
| DataBaseId | 表ID | short |
| TableId | 表ID | short |
| FieldId | 字段ID | int |
| RecordCount | 存储记录数 | long |

（5）分布式文件系统分配表

分布式文件系统分配表（Distributed File System Allocate Table，DFS\_AT）主要管理集群中所有的DB当前的存储状态信息。

|  |  |  |
| --- | --- | --- |
| 变量名 | 含义 | 类型 |
| NodeId | 节点ID | short |
| DBIndex | 所在节点序号 | short |
| Flag | 是否存满 | short |
| OffSet | 当前存储条数 | long |

2.2.2 Nanomsg传输协议

在本次课题研究工作中，节点之间的控制信息以及文件传输都是用了Nanomsg传输协议。Nanomsg是由Martin Sustrik实现的一个可靠通信库，采用C语言实现，几乎不需要任何依赖，且使用轻便，适用于多种操作系统。它提供了几种常见的通信模式，根据实际需要进行组合，可为用户实现各类通信需求。目前，Nanomsg可用的通信模式主要有：

1. 配对模式（PAIR）：简单的一对一通信模式；
2. 总线模式（BUS）：多对多通信；
3. 请求回复模式（REQREP）：支持组建大规模的集群服务来处理用户请求；
4. 扇入扇出模式（PIPELINE）：支持多对一或一对多的通信方式；
5. 订阅模式（PUBSUB）：将消息发送至订阅消息的用户；
6. 调查模式（SURVEY）：在单一请求中查询多个应用程序的状态。

另外，Nanomsg不仅支持多种传输模式，还支持以下多种传输机制：

1. INPROC：单进程内通信；
2. IPC：单机内多进程的通信；
3. TCP：通过tcp协议的网络通信。

基于Nanomsg的这些优点，本课题在仿真实验中进行控制信息和数据文件传输时，均采用了该协议。

2.2.3数据备份策略

分布式存储系统中数据被切分为固定大小的数据块（DataBlock，DB）存储在这些节点上。在集群存储中，常见的备份方法有直接复制、XOR（异或）等策略[14]。关于备份策略的选择，本课题做了以下研究：

1）复制备份策略是分布式系统中最常用的备份策略之一。比如在HDFS中，一般都是采用三副本存储策略。该策略实现简单、可靠性高，只有整个集群中同时有三台节点损坏时，数据才有可能丢失。假设集群中有n台节点，数据备份为m，每台节点宕机概率为p，在忽略其他故障因素的情况下，集群中可能出现数据损坏的概率为：

2）XOR策略采用了异或运算实现对数据块的备份。该策略将存储数据分为m份，将这些数据直接异或运算生成一块校验数据块，系统将这些数据块分别存储到不同的节点上。当其中任一台节点宕机时，总能通过剩余的其他数据块异或运算进行修复。但是，当集群中任两台节点故障时，存储数据很有可能损坏。假设集群节点数目为n，每份数据被分为m块，异或计算得到1个校验数据块，每台节点宕机的概率为p，在忽略其他故障因素的前提下，集群中可能出现数据损坏现象的概率为：

由公式2-1和2-2可知，即直接复制的存储可靠性较高。由于本次研究以尽可能高的存储可靠性为目标，所以在研究工作中将选择直接复制的方法作为副本存储策略。

2.2.4 HDFS中存储调度算法

目前，HDFS分布式存储策略凭借其优秀的存储机制，成为了大数据存储中最受人们欢迎的系统之一[16]。本课题详细分析了HDFS中分布式存储策略机制，并针对其负载均衡方面提出了一种基于字段均衡的分布式存储算法，分别对两种存储机制进行试验仿真，对结果进行讨论分析。

在HDFS的分布式存储调度算法中，采用了直接复制的备份策略。以默认的三副本存储为例，其存储调度策略算法步骤大致如下：

1. 数据上传节点将所上传数据的信息发送到集群管理节点；
2. 管理节点检查数据上传节点是否在集群内，如果是集群内某一节点且其存储空间合理，该数据上传节点将会直接被选择为副本一存储节点；否则，从集群内随机选择一台节点作为副本一存储节点；
3. 在除副本一存储节点所在的机架外，随机选择一个机架中的两台不同节点，分别作为副本二和副本三的存储节点；
4. 管理节点将所有副本的存储位置信息发送到数据上传节点，数据上传节点将DB以及副本位置信息发送到副本一存储节点，并存储DB为副本一；
5. 副本一存储完成之后，所在节点通知管理节点更新存储信息，然后将DB以及副本三的存储位置信息发送到副本二存储节点；
6. 副本二存储完成之后，通知管理节点更新存储信息，然后将DB发送到副本三存储节点；
7. 副本三存储完成之后，通知管理节点更新存储信息；
8. 数据存储完成。

2.3负载均衡技术

2.3.1负载均衡的类别和意义

负载均衡的核心就是各节点“分摊压力”。分布式存储系统的负载均衡是一种动态均衡技术，主要工作是把分布式系统中各节点上存储的数据均衡分配。在分布式系统中，常见的负载均衡有数据存储时的负载均衡以及集群扩容后的负载均衡。

（1）数据存储时负载均衡

数据存储时的负载均衡主要由集群中Master节点通过实时读取元数据的信息，分析集群中各节点当前的存储状况，选择存储数据量较少的节点存储当前数据块，从而达到各节点存储量相对均衡的状态，使得集群中每台节点的使用发挥到最大限度。最后，当每台节点的数据存储容量都达到饱和时，就必须通过增加物理服务器数量来解决这个问题。此时，就需要用到集群扩容后的负载均衡技术使新增节点和集群中原节点的存储容量达到新的平衡状态。

（2）集群扩容后负载均衡

集群扩容后，新增节点上没有存储数据，不能参与任务调度。负载均衡就是把原节点上的部分数据迁移至新增节点上的过程。所以，集群负载均衡在某种程度上拓展了服务器带宽，增加了集群吞吐量，提高了系统数据处理能力。解决了集群中网络拥塞问题，为实现就近提供服务和节点位置无关分布提供了保障。

总的来说，负载均衡的主要意义有：

1. 为用户提供更好的访问质量；
2. 提高服务器响应速度；
3. 提高了服务器和其他资源的利用效率；
4. 缓解了集群网络中关键节点失效带来的问题。

2.3.2负载均衡算法

目前，在分布式系统中负载均衡算法多种多样，其中比较常见的主要有直接轮循、加权轮循、固定权重、最少连接数、加权最少连接、源IP哈希等算法。

（1）直接轮循算法

轮循算法按照先后顺序把客户端发来的请求依次分发给每一个服务器节点，使每台节点任务量相对均衡[30]。比如，在分布式系统中，数据存储调度时Master节点把每个数据块请求按照先后顺序分发给各个节点，使得每台节点上的存储容量相对均等。轮循算法在发部分情况下都工作的不错，尤其是各服务器节点在处理速度和内存等方面相对均等的时候，工作效果会更加令人满意。

（2）加权轮循算法

加权轮循算法与直接轮循算法的区别就在于该算法考虑到集群中每台服务器节点可能性能差别较大。假如在同一集群中，有的节点性能远远高于其他节点，为了提高整体集群的工作效率，应为高性能的节点分配更多的任务[17]。管理员只要根据节点性能进行权重分配，比如为A节点分配权重值2，为B节点分配权重值1，表示轮循一次，A节点接收2个客户端请求的同时B节点接收一个请求，以此类推。加权轮循算法比较适合集群中节点性能参差不齐的状况。

（3）固定权重算法

固定权重只有在其他服务器的权重值都很低时才使用。然而，如果最高权重服务器的权值减小，则下一个最高优先级的服务器将为客户端服务。这种方式中每个真实服务器的权重需要基于服务器优先级来配置。这种负载均衡算法更适合在数据访问时使用，比如用户需要查找某些数据块，集群从权重最高的节点开始查找，也就是从性能最高的节点开始到性能低的节点，直到查找到相应的副本为止。这样保证了用户所需数据块将尽可能由性能较高的节点提供，在一定程度上保证了集群性能[35]。

（4）最少连接数算法

轮循和加权轮循算法都没有考虑到系统是否识别在一定的时间内各节点与客户端保持了多少请求连接。所以很有可能发生服务器A接收请求多于服务器B，但是A节点上请求大都处理完成，B节点上的请求都还在处理中。此时如果按照轮循算法，服务器B将会优先被分配请求，导致其任务压力更大[36]。针对集群中这种潜在的问题，可以通过最少连接数算法解决。当为集群中服务器节点分配请求时，比较每台节点当时所打开的连接数，优先将请求分配给连接数最少的服务器节点。该算法比较适合请求繁忙的集群，集群中每台节点都有若干未完成的请求，将新来请求分配至连接最少的节点上。如果客户端请求很少，仅几台节点就可以处理。集群中大部分节点不会因为请求过多累计，所以每次比较得出的连接数较少的节点总是集群中靠前的某一台，最后导致请求任务都由这几台节点处理。这种情况如果发生在数据存储调度中，会引起数据大都累积在个别节点上，导致集群存储严重失衡的后果。

（5）加权最少连接算法

加权最少连接算法结合了权重和最少连接算法这两者优势，由管理员根据服务器的情况定制权重计算出每台节点的最少连接数，这样达到了对服务器更加平衡的应用。结合实际连接数和权重值计算出每台节点的理论连接数，把请求分发给理论连接数最少的节点[38]。这种算法同样适合于请求比较繁忙的集群，在任务清闲的集群中应用该算法同样会导致任务请求分配严重不均的后果。

（6）源地址哈希算法

源地址哈希的思想是根据获取客户端的IP地址，通过哈希函数计算得到的一个数值，用该数值对服务器节点总数进行取模运算，所得结果便是客户端要访问服务器的序号。采用源地址哈希法进行负载均衡，同一IP地址的客户端，当后端服务器列表不变时，它每次都会映射到同一台后端服务器进行访问。不过这种均衡算法适合客户端请求量相对均衡的情况，如果客户请求中，某些IP请求量远远高于其他IP，则最后的均衡结果将会很不理想[39]。

第三章 基于字段均衡的分布式存储调度算法

3.1基于字段均衡的分布式存储算法

3.1.1HDFS中分布式存储调度研究

HDFS是目前分布式系统中应用最广泛的文件系统之一，它不仅具有高容错性等特点，还很适合用来部署在低廉的硬件上。该文件系统适合超大数据集的应用程序，可实现流的形式访问数据。在HDFS分布式存储调度算法中考虑了数据存储时在各节点上的负载均衡问题，目的是使数据在各节点上存储更加均衡，从而提高集群任务调度效率。



图3-1 HDFS中分布式存储调度流程图

在对于列数数据而言，HDFS分布式存储调度策略虽然能够基本保证各节点在存储空间上相对均衡，但是字段在各节点上的存储却很可能偏颇较多。如图3-1中，按照HDFS分布式存储调度算法，有可能会存储这样的结果。在三台节点的集群中，存储A、B、C三种字段的数据，由于该存储算法在选择数据块存储位置时主要根据节点当前存储空间，并没有考虑各字段在节点上的分布，所以在某些情况下，可能会导致节点之间存储总量基本均衡，但是各字段在节点之间的存储差别较大。如图3-2所示，A、B、C三种字段的数据块一共有27块，三台节点上每台均存储了9个数据块，整体存储均衡。但是三种字段在每台节点上的分布不均，A字段大部分都存储在1号节点上，B字段大部分存储在2号节点上，C字段全部存储在3号节点上，各节点DB存储数量如表3-1所示。在列式数据库中，数据调度任务一般都是基于字段的，假设图3-2中的C字段为某学校某年级学生的数学成绩，现先要计算概念机所有学生数学课平均成绩，则在数据库中调度C字段进行计算即可。对于分布式数据库而言，越多节点参与任务调度越有助于提高任务计算效率，但是在图3-2所示的情况中，C字段全部存放在3号节点上，所以只有3号节点参与该任务计算。假如C字段在各节点上均匀分布，在三台节点性能相同的情况下，这次所有学生数学成绩平均值的计算效率应该是当前的三倍。



图3-2 HDFS中字段存储不均示例

表3-1 三台节点上各字段DB存储上数量

|  |  |  |  |
| --- | --- | --- | --- |
| 节点  DB数量  字段 | 节点1 | 节点2 | 节点3 |
| A | 8 | 4 | 0 |
| B | 1 | 5 | 3 |
| C | 0 | 0 | 2 |

3.1.2基于字段均衡的分布式存储算法

基于HDFS分布式存储调度算法中所存在的字段在各节点存储不均衡的情况，本文提出了基于字段均衡的分布式存储调度算法。

在基于字段均衡的分布式存储算法中，以三副本复制的备份策略为例，其存储调度算法步骤大致如下：

（1）数据库引擎启动时，管理节点从本地读取集群中所有数据块的存储信息；

（2）管理节点收到客户端上传数据块的信息后，根据该数据块的字段ID，查找内存中对应字段已存储的上一个DB信息，如果还未存满，则该DB即为正本数据的优先存储位置；如果该DB剩余空间不足以存储用户上传数据块，则管理节点再遍历该字段所有DB信息，统计其在集群中每个节点的DB数量，选取该字段DB数量最少的节点作为副本一存储节点；

（3）除去与副本一节点所在相同交换机下的节点，重复步骤1)和2)，选取该字段最少的一台节点作为副本二的存储节点；

（4）除去与副本一以及副本二存储节点所在相同交换机下的节点，重复步骤1)和2)，选取该字段最少的一台节点作为副本三的存储节点；

（5）管理节点将数据块的所有副本存储位置信息发送到用户上传数据的节点，该节点将副本二和副本三的存储位置信息以及数据块发送到副本一存储节点；

（6）副本一接收存储完成之后，通知管理节点更新相关存储信息，然后将副本三的存储位置信息以及数据块发送到副本二存储节点；

（7）副本二接收存储完成之后，通知管理节点更新相关存储信息，然后数据块发送到副本三存储节点；

（8）副本三接收存储完成之后，通知管理节点更新相关存储信息；

（9）数据存储完成。

具体流程图如图3-3所示。



图3-3 基于字段均衡的分布式存储调度算法流程图

3.2字段均衡性

在图3-2中所示的HDFS中可能出现数据整体在各节点存储均衡，但是各字段在节点上不均衡的情况，这样在集群中不能充分利用每台节点的计算资源。在基于字段均衡的分布式存储算法中，不仅使得各节点上的数据存储量基本达到均衡，还能使得各字段基于节点均衡。如图3-4所示，在该算法中，每个字段的数据块存储时，集群中Master节点都会比较当前各节点中该字段已存储数据块个数，选取该字段数据块存储较少的节点作为存储节点。所以最后能够使得每个字段在各节点的存储达到基本均衡。每台节点上的A、B、C三种字段的数据块存储数量如表3-2所示。

另外，当某字段需要存储的数据块数量不能被节点数整除时，每台节点上该字段的数据块数量虽然不完全相等，假如字段数据块总数除以节点数余数为m，由于基于字段均衡的分布式存储调度算法基于节点轮循，则集群中交换机下的前m台节点上的该字段数据块数量比其他节点上多一个。如表3-2中C字段在1、2、3号节点上的存储数量分别为2、2、1，前面两台节点上C字段数据块数量比3号节点多1个，如果之后客户端继续上传C字段数据，Master节点通过比较当前各节点上C字段数据块的存储数量可知3号节点最少，选择3号节点为存储节点。



图3-4 各节点上基于字段均衡的分布式存储示例

表3-2 各节点上DB存储个数

|  |  |  |  |
| --- | --- | --- | --- |
| 节点  DB数  字段 | 节点1 | 节点2 | 节点3 |
| A | 4 | 4 | 4 |
| B | 3 | 3 | 3 |
| C | 2 | 2 | 1 |

3.3存储可靠性

3.3.1集群稳定性

如图3-5所示，在基于字段均衡的分布式存储策略中，某DB的三份副本分别存储在1号、2号和r号交换机下的节点上。当集群中交换机均正常工作的情况下，集群中节点故障数量不超过两台时，才能保证该DB数据的信息完整性；当其中任意一台或两台交换机故障时，仍然有DB副本可访问，只有在1号、2号和r号交换机都发生故障时，该DB数据完全丢失，即在跨交换机的分布式存储调度算法中，集群最多容忍两台交换机发生故障。



图3-5 基于字段均衡的分布式存储示例

因此，在跨交换机的分布式存储调度策略中，DB的每份副本都存储在不同交换机下的节点上，所以在保证所有DB信息可访问的情况下，集群最多可容忍m-1台交换机故障，此时交换机故障容忍率为：

当交换机都正常工作时，整个集群节点故障容忍率为：

在HDFS分布式存储策略中，当副本数大于三时，前三份副本存储在两台交换机下的三台节点上，剩余的副本将均匀地存储在其他交换机下。表3-3展示了HDFS中和跨交换机的分布式存储调度策略中集群对节点和交换机的故障容忍度：

表3-3 不同存储方法下数据可靠性能

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 性能指标  存储  调度算法 | 交换机故障容忍数量（台） | 集群中节点故障容忍数量（台） | 交换机故障容忍率 | 集群中节点故障容忍率 |
| 基于字段均衡的存储调度算法 | m-1 | m-1 | (m-1)/r | (m-1)/(nr) |
| HDFS存储调度算法 | m-2 | m-1 | (m-2)/r | (m-1)/(nr) |

在交换机都正常工作的情况下，从节点故障的角度而言，两种调度策略的节点故障容忍度相同；从交换机故障容忍度而言，跨交换机的分布式存储调度策略比HDFS中分布式存储调度策略多1。对于人们最常用的三副本存储策略，假设节点正常工作，从交换机故障的情况来说，基于字段均衡的分布式存储策略的存储可靠性是HDFS中分布式存储策略存储的2倍。所以，相比于HDFS中存储调度算法，跨交换机的存储调度算法的数据存储可靠性更高。

3.3.2数据故障处理

基于字段均衡的分布式存储策略中已经采用了跨交换机存储DB备份的方法，就是为了防止部分DB损坏或节点故障时存储数据丢失。这在一定程度上提高了数据在集群中存储可靠性，但是数据丢失或损坏的问题是不可能杜绝的，比如磁盘损坏、人为误删数据等。为了最大限度地保障用户对数据库可用， 分布式文件系统应该能够应对一定程度上的数据故障。

3.3.2.1 DB故障

如图3-6所示，展示了当集群中节点正常工作但是部分DB丢失时的数据访问流程。由于节点正常工作，所以Master节点不能在第一时间发现其中某些DB丢失。当客户端向Master节点发出DB读取请求时，Master节点首先查询本地的DB存储信息表，获取相应的DB副本位置并返回给客户端。然后客户端向其中一个副本存储节点发送读取数据的请求，发现DB丢失时，节点将故障信息报告给Master节点，继续读取下一个副本，直到成功读取数据或者遍历所有副本。



图3-6 分布式数据库中DB读取流程图

（1）数据修复

如表3-4，假设在9台节点的集群规模中，1~3、4~6、7~9号节点分别部署在三台不同的交换机下，字母A~I表示不同的DB，相同字母的不同下标表示不同副本。假设集群中节点均正常工作，但是2号节点中数据块B1被已误删，当客户端向Master节点发送数据块B的访问请求时，Master查询DB存储信息表，获取2、5、8号节点的存储为位置返回给客户端。首先客户端向2号节点查询数据块B1，由于该DB已被删除，此时2号节点将该DB丢失信息报告给Master节点，Master节点查询DB存储信息获得数据块B的其他副本位置，即5号节点上的B2，并向5号节点下发指令复制数据块B，存储在故障副本所在的节点上，即2号节点，并在存储信息表中的原B1的存储信息删除。副本B1修复完成之后的存储完成状态如表3-5所示。

表3-4 DB故障时示例

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| A1 | ~~B1~~ | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 |
| D2 | E1 | F1 | D3 | E2 | F2 | E3 | G1 | F3 | G2 |
| G3 | H1 | I1 | H2 | I2 | J1 | H3 | I3 | J2 | J3 |

表3-5 DB故障数据修复示例

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| A1 | ~~B1~~ | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 |
| D2 | E1 | F1 | D3 | E2 | F2 | E3 | G1 | F3 | G2 |
| G3 | H1 | I1 | H2 | I2 | J1 | H3 | I3 | J2 | J3 |
|  | B1 |  |  |  |  |  |  |  |  |

（2）数据存储

在某些DB故障，节点仍然能够正常工作时，系统会及时重新复制发生故障的DB其他副本进行修复，对于之后客户端上传的其他DB，仍然按照基于字段均衡的分布式存储调度算法进行存储。

以表3-5的存储状态为例，假设客户端新上传了数据块K并存储三份副本，根据每台节点上已存储的该字段的数量，K1、K2、K3分别存储在1号、4号和7号节点上，如表3-6所示。

表3-6 数据修复后数据存储示例

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| A1 | ~~B1~~ | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 |
| D2 | E1 | F1 | D3 | E2 | F2 | E3 | G1 | F3 | G2 |
| G3 | H1 | I1 | H2 | I2 | J1 | H3 | I3 | J2 | J3 |
| K1 | B1 |  | K2 |  |  | K3 |  |  |  |

3.3.2.2节点故障

当集群中节点上有DB丢失或损坏时，只影响了部分DB的读取，与其他DB的存储无关。但是，当集群中出现整个节点宕机或者一个交换机故障时，不仅会影响到大量数据的读取，还会给之后上传数据的存储造成不便。

（1）数据修复

如表3-7所示，10台节点规模地集群已经基于字段均衡存储，如果其中一台节点发生故障，以5号节点为例，此时Master通过心跳机制发现节点5发生故障，并将其记录为故障节点。为了使集群中每个DB遵循三副本存储，系统需要对5号节点上的DB副本进行修复。

Master节点通过DB存储信息表，查询该故障节点上存储DB，即B2、E2、I2。然后Master在分别查询数据块B、E和I的其他副本存储位置，即数据块B1、E1存储在2号节点，I1存储在3号节点，E3存储在7号节点，B3、I3存储在8号节点，Master节点下发指令给2号和3号节点并复制数据块B、E和I，分别对其以基于字段均衡的分布式存储策略进行存储。存储完成后的结果如表3-8所示。

表3-7 节点故障示例

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| A1 | B1 | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 |
| D2 | E1 | F1 | D3 | E2 | F2 | E3 | G1 | F3 | G2 |
| G3 | H1 | I1 | H2 | I2 | J1 | H3 | I3 | J2 | J3 |

表3-8 节点故障后数据修复示例

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| A1 | B1 | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 |
| D2 | E1 | F1 | D3 | E2 | F2 | E3 | G1 | F3 | G2 |
| G3 | H1 | I1 | H2 | I2 | J1 | H3 | I3 | J2 | J3 |
|  |  |  | B2 |  | E2 |  |  |  | I2 |

（2）数据存储

在集群中某节点故障之后，系统会迅速地恢复故障节点上的DB副本。之后，系统在进行数据存储时，该节点在修复之前暂不参与。在如表3-8中存储状态的基础上，假如客户端继续上传数据块K，存储三份副本，按照基于字段均衡的跨交换机分布式存储策略，如表3-9所示，K1存储在1号交换机下的1号节点，K2存储在2号交换机下的4号节点，K3存储在3号交换机下的7号节点。

表3-9 数据修复后数据存储示例

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| A1 | B1 | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 |
| D2 | E1 | F1 | D3 | E2 | F2 | E3 | G1 | F3 | G2 |
| G3 | H1 | I1 | H2 | I2 | J1 | H3 | I3 | J2 | J3 |
| K1 |  |  | B2 |  | E2 | K3 |  |  | I2 |
|  |  |  | K2 |  |  |  |  |  |  |

（3）节点修复

当故障节点修复之后，加上该节点上存储的DB副本，集群中这些副本的数量超过了规定数量，所以需要删除部分多余的副本。Master节点查询故障节点上的DB信息，让后分别向与该节点上重复的DB所在节点下发指令进行删除操作。以表3-9中的存储为例，当5号节点恢复工作之后，Master查询DB存储信息，获得5号节点上与数据块B2、E2、I2重复的节点存储位置，并向其所在节点4号、6号和10号节点下发删除指令进行删除操作。删除完成之后存储状态如表3-10所示。

表3-10 节点修复后多余副本的删除操作示例

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| A1 | B1 | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 |
| D2 | E1 | F1 | D3 | E2 | F2 | E3 | G1 | F3 | G2 |
| G3 | H1 | I1 | H2 | I2 | J1 | H3 | I3 | J2 | J3 |
| K1 |  |  |  |  |  | K3 |  |  |  |
|  |  |  | K2 |  |  |  |  |  |  |

3.3.2.2交换机故障

由于本章提出的基于字段均衡的分布式存储算法中各DB是跨交换机存储的，所以当集群中发生故障的交换机数量少于备份数量时，仍然能够保证集群继续工作。如表3-11所示，以集群中共有四台交换机为例，每台交换机下分别有三台或两台节点。表3-11中各副本已经基于字段均衡存储完成，如果当集群中某台交换机故障时，以1号交换机故障为例，此时Master节点和该交换机下的1、2、3号节点失去联系，将它们记录为故障节点。

表3-11 交换机故障示例

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| A1 | B1 | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 | E1 |
| D2 | E2 | F1 | D3 | E3 | F2 | F3 | G1 | H1 | G2 | H2 |
| G3 | H3 | I1 | I2 | J1 | K1 | I3 | J2 | K2 | J3 | K3 |

（1）数据修复

当Master节点将1号交换机下的三台节点都记为故障节点之后，就开始对这些节点上的DB副本进行修复。Master节点查询存储信息表，获得故障节点上存储的DB，然后查询其他副本的位置。以轮循的顺序，按照一个DB的不同副本存储在不同交换机下的原则，从最近的一个副本进行复制存储。以A1为例，Master查找到其他副本分别存储在4号和7号节点上。Master节点对4号节点下发指令复制A2至4号交换机下的10号节点，副本名称取为A1，保证了数据块A的三份副本仍然位于不同的交换机下。对于故障节点上的其他DB，也是同样的步骤进行修复。修复完成之后的各DB存储位置如表3-12所示。

表3-12 交换机故障后数据修复示例

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| A1 | B1 | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 | E1 |
| D2 | E2 | F1 | D3 | E3 | F2 | F3 | G1 | H1 | G2 | H2 |
| G3 | H3 | I1 | I2 | J1 | K1 | I3 | J2 | K2 | J3 | K3 |
|  |  |  | G3 | H3 |  | D2 | E2 |  | A1 | B1 |
|  |  |  |  |  |  |  |  |  | C1 | F1 |
|  |  |  |  |  |  |  |  |  | I1 |  |

（2）数据存储

在集群中某交换机故障之后，系统会迅速地恢复故障节点上的DB副本。之后，系统仍然会进行正常的数据存储工作，不过在故障交换机修复之前，其部署在该交换机下的节点一直都会被视为故障节点，所以不参与数据存储工作。以表3-12中存储状态为例，客户端依次上传L、M两个DB，每个DB三份副本，按照基于字段均衡的跨交换机分布式存储策略，L1、L2、L3分别被存储在6号、9号和11号节点，如表3-13所示。

表3-13 交换机故障后数据存储示例

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| A1 | B1 | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 | E1 |
| D2 | E2 | F1 | D3 | E3 | F2 | F3 | G1 | H1 | G2 | H2 |
| G3 | H3 | I1 | I2 | J1 | K1 | I3 | J2 | K2 | J3 | K3 |
|  |  |  | G3 | H3 | L1 | D2 | E2 | L2 | A1 | B1 |
|  |  |  |  |  |  |  |  |  | C1 | F1 |
|  |  |  |  |  |  |  |  |  | I1 | L3 |

（3）故障修复

在交换机故障后，虽然系统已经将数据存储调整为均衡状态，但是在交换机修复之后，该交换机下节点上的DB副本就比规定副本数量多了一个。所以Master节点将会把数据修复时存储的DB进行删除。以表3-13存储状态为例，删除之后的存储如表3-14所示。

表3-14 交换机修复后多余副本删除操作示例

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| A1 | B1 | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 | E1 |
| D2 | E2 | F1 | D3 | E3 | F2 | F3 | G1 | H1 | G2 | H2 |
| G3 | H3 | I1 | I2 | J1 | K1 | I3 | J2 | K2 | J3 | K3 |
|  |  |  |  |  | L1 |  |  | L2 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | L3 |

3.4数据存储与访问的效率

在HDFS和跨交换机的分布式存储调度策略中，一个DB的副本都是依次存储的。如图3-7，DB首先存储为副本一，然后副本一传输DB信息至第二台目的节点存储为副本二，最后副本二再传输DB信息到第三目的节点存储为副本三。在跨交换机的分布式存储调度中，也是如此。因为在同一交换机下的节点之间文件传输速度比不同交换机下的节点之间文件传输速度快，所以对于一个DB的所有副本的存储，HDFS中分布式存储比跨节点的分布式存储效率高。但是，在副本二节点向副本三节点进行DB传输时，客户端已经完成了向集群传输副本一的任务，并且已经开始了其他DB的副本一写入。假设在跨交换机的分布式存储策略中存储一个DB所需存储时间比在HDFS分布式存储策略中所需时间多△t，当集群连续存储多个DB时，由于各台节点的存储工作是并行的，所以随着存储DB数量的增长，相同数量的数据在两种分布式存储系统中的存储时间差总是为一个DB的存储的时间差△t。



图3-7 HDFS和跨节点的分布式存储时间对比

另外，对于分布式存储系统中的数据，一般都遵循“一次写，多次读”的原则，而且集群中DB的访问效率与任务调度效率密切相关。当管理节点对某DB进行调度时，只要查询并访问到距离最近的该DB的一个副本即可。以三副本为例，在HDFS分布式存储策略中，每个DB的副本分布在两台交换机下的节点上，跨交换机的分布式存储策略中每个DB的副本分布在三台不同的交换机下，即跨交换机的分布式存储策略中每个DB的副本存储更加分散均衡，所以管理节点与每个DB的最近副本的平均距离也相对较短，所以跨交换机的分布式存储策略更有助于提高集群任务调度效率。

3.5存储堆积问题及修正

3.5.1部分字段存储堆积问题

基于字段均衡的分布式存储调度算法在大分部分情况下可使得每个字段的数据在集群中各节点上存储均衡，相应的每台节点上的数据存储总量也基本相等。其原理是集群中Master节点在误差为一的范围内对每个字段基于节点进行轮循，可是这种轮循方法在客户端上传多个不同字段的单个数据块时，会引起大量不同字段的数据块堆积在集群中靠前的节点上。如图3-8所示，A字段的数据块总数比节点数的整数倍多1，在每台节点均存储一个A字段数据块之后，剩余的一个数据块被优先分配在第一台节点上。在极端情况下，如果不断有这种新的字段在该集群存储，会导致集群中越靠前的节点存储数据量越大，造成各节点上数据量存储严重不均的后果。



图3-8 数据存储堆积问题示例

3.5.2算法优化

系统在为DB选择存储节点时，整体上按照一定的点顺序轮循的机制，所以在轮循顺序列表中前面的节点存储DB数量总大于或等于后面节点上DB的存储数量。所以如果能够使系统在计算DB存储位置时，保证集群中第一台节点1和最后一台节点n之间的存储容量差保持在一定的范围内，即可解决多个不同字段在集群部分节点的堆积问题。

对于多个不同数据块在部分节点上存储堆积的问题，在基于字段均衡存储中，比较各节点上某字段已存储数据块个数时，要求在满足单个字段基于节点均衡的基础上还必须满足每台节点上DB数量与集群中最后一台节点上DB数量相差不超过某定值η。由于每个DB的大小为256M，所以，利用优化后的基于字段均衡的分布式存储策略可使得集群中各节点上的数据存储最多相差不超过η个DB。η默认值为10，这对于普通容量的磁盘来说，已经比较理想了。用户可根据实际需求自己设定η的值，范围为5~20。η值越小，这种大量不同字段存储时各节点之间的均衡性就越强。即集群中第一台节点与最后一台节点之间的最大存储容量差S如公式3-3所示。

改进后的算法步骤为：

1) 数据库引擎启动时，管理节点从本地读取集群中所有数据块的存储信息；

2) 管理节点收到客户端上传数据块的信息后，根据该数据块的字段ID，查找内存中对应字段已存储的上一个DB信息，如果还未存满，则该DB即为正本数据的优先存储位置；如果该DB剩余空间不足以存储用户上传数据块，则管理节点统计其在集群中每个节点上该字段的DB数量，从1号节点轮循遍历，选取该字段DB数量最少并且与集群中最后一台节点上DB数相差不超过η的节点作为副本一存储节点；

3) 除去与副本一节点所在相同交换机下的节点，选取该字段DB数量最少并且与集群中最后一台节点上DB数相差不超过η的节点作为副本二存储节点；

4) 除去与副本一以及副本二存储节点所在相同交换机下的节点，选取该字段DB数量最少并且与集群中最后一台节点上DB数相差不超过η的节点作为副本三存储节点；

5) 管理节点将数据块的所有副本存储位置信息发送到用户上传数据的节点，该节点将副本二和副本三的存储位置信息以及数据块发送到副本一存储节点；

6) 副本一接收存储完成之后，通知管理节点更新相关存储信息，然后将副本三的存储位置信息以及数据块发送到副本二存储节点；

7) 副本二接收存储完成之后，通知管理节点更新相关存储信息，然后数据块发送到副本三存储节点；

8) 副本三接收存储完成之后，通知管理节点更新相关存储信息；

9) 数据存储完成。



图3-9 算法优化后字段存储示例

以集群中前部分节点中的存储情况为研究对象，且它们均分布在不同交换机下，假设集群中每个DB存储三份副本，其中A、B、C、D等均代表不同的字段，不同的下标如A1、A2、A3代表同一DB的不同副本。理想情况下，在优化后的基于字段均衡的分布式存储调度算法中，集群中每台节点上的DB数量相比最后一台节点上DB数量不多于η。当η=5时，如图3-9所示，假设客户端上传的数据中，每个字段仅有一个DB，每个DB存储三份副本。初始集群中节点均为空，开始存储的A、B、C、D、E字段均选择从1号节点开始存储，当开始存储F字段时，虽然每台节点上的F字段DB数量均为零，但是由于节点1、2、3上DB数量均为5个，最后一台节点n上DB数量为零，相差5个DB，此时系统会继续轮循查找F字段DB数量最少并且节点上DB总数量与节点n上DB数量0相差少于5的节点，即节点4作为F字段的副本一存储位置。在F字段另外两个副本存储完成之后，集群继续处理G字段的存储请求。此时G字段为新字段，每台节点上该字段DB数量均为零，但是1、2、3号节点上数据块总量均为5，集群中最后一台节点n上DB数量仍为零，相差5个DB，所以系统继续轮循至4号节点，由于此时节点4上DB数量为1，与节点n上DB数量相差仅1个DB，所以节点4被选为字段G副本一的存储节点。同理，字段H也是这样被存储在了节点4上。

3.6小结

本章首先研究了HDFS中分布式存储调度算法，针对该算法中每个字段数据在各节点之间存储不均衡的问题进行分析，提出了基于字段均衡的跨交换机分布式存储策略，使得每一字段的数据在集群中各节点上的存储DB数量达到基本均衡，并且使同一DB的不同副本分布在不同交换机下的节点上，提高了数据存储的可靠性。

基于字段均衡的分布式存储算法与HDFS中分布式存储算法主要区别在于选择DB存储位置时，系统会计算当前每台节点上该字段的DB数量，选取合适的数量最少的节点作为目的节点，以此保障同一字段数据均衡存储在不同的节点上。之后本章通过理论分析及仿真证明了这种基于字段均衡的分布式存储调度算法不仅使得各字段数据在节点之间存储均衡，还提高了集群中数据存储的可靠性。

考虑到一些极端情况的存在，比如当大量不同字段的单个DB进行存储时会堆积在集群中前面的几台节点，本章对提出的算法进行了优化。在选择当前DB的存储位置时，要求被选择的节点上存储的所有字段DB数量不能比集群中最后一台节点上的DB数量相差太多。通过仿真证明了改进后的基于字段均衡的跨交换机分布式存储调度算法保证了各字段DB数量以及数据总DB数在集群中各节点上的均衡。

第四章 新增节点后的负载均衡策略

4.1算法步骤

基于字段均衡的跨交换机分布式存储调度算法，保证不仅是每个字段数据在节点之间存储分布基本达到均衡状态，还保证了每个DB的不同副本存储在不同的交换机下，提高了数据存储的可靠性。集群在增加节点后，为了使得集群中数据在每台节点上的存储相对均衡，系统会按照一种算法将原来节点上的已存储的部分DB迁移到新增节点上去。该算法必须使集群中数据仍旧保证以下几点：

1）各节点存储容量基本均衡；

2）每个字段数据基于节点存储均衡；

3）每个DB的不同副本仍然跨交换机存储。

负载均衡主要工作就是DB在节点之间的传输与磁盘读写，同时，这些工作在负载均衡时也最消耗时间。所以在负载均衡过程中，应尽量避免不必要的DB传输与读写，从而提高均衡效率。另外，集群新增节点有多种情况，可能新增节点部署在已有交换机下，或者新增节点规模较大，集群新增了交换机，该交换机下新增了其他节点。集群增加节点越多，需要迁移的DB就越多，花费时间也就越久。

新增节点后，具体的负载均衡算法步骤如下：

（1）系统读取新增节点交换机个数以及新增节点个数；

（2）Master节点根据数据库中某个字段DB数量和节点台数，计算得出平均每台节点上应该存储的DB数量n，则应该移动到每台新增节点上n个DB，原有节点上的DB数量减去n即需要从原有节点上移走的DB数量；

（3）系统读取新增节点所在交换机下的所有节点中存储的该字段DB信息；

（4）集群以第一台空的节点作为目的节点，分别遍历原有的节点作为源节点，将新增节点所在交换机下没有副本的部分DB复制发送到目的节点，同时为保证每个DB的不同副本不被复制到同一个交换机下，同时记录新增交换机下节点中移过来的DB副本信息；

（5）复制完成后，删除原有节点上移动的DB，并更新Master节点上存储信息表；

（6）分别以其他新增节点为目的节点，重复步骤4）和5），直到所有新增节点都被作为目的节点接收DB均衡操作；

（7）迭代步骤2）~6），分别对其他字段进行同样的均衡操作，直到所有字段都完成负载均衡；

（8）负载均衡结束。

4.2负载均衡分类

4.2.1新增节点部署在原交换机下

既然集群中会增加节点，那么就要部署新增节点的部署位置。根据实际情况，当集群中的某些交换机下还有空闲位置时，尽量将新增节点部署在原有的交换机下，否则需要同时在集群中增加交换机来管理节点。

如表4-1和4-2所示，分别示例了将新增节点部署在原有交换机下时的负载均衡过程。如表4-1，表格中第一行表示每台节点的序号，集群中原有10台节点，每连续三台节点部署在同一台交换机下，即1~3、4~6、7~9、10号节点分在四台不同的交换机下。表格中的大写字母代表某一字段数据的不同DB，数字下标表示不同副本。在这10台节点的集群中，存储A~J 10个不同的DB，每个DB三份副本，且每个DB的不同副本存储在不同的交换机下。为了扩大集群容量，先增加两台节点到10号节点所在的交换机，此时数据存储现状如表4-1所示。

表4-1 新增节点部署在原有交换机下的负载均衡前存储状态

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| A1 | B1 | C1 | A2 | B2 | C2 | A3 | B3 | C3 | D1 |  |  |
| D2 | E1 | F1 | D3 | E2 | F2 | E3 | G1 | F3 | G2 |  |  |
| G3 | H1 | I1 | H2 | I2 | J1 | H3 | I3 | J2 | J3 |  |  |

按照本章提出的新增节点后的负载均衡算法，系统首先计算节点原来存储该字段的所有DB数量为30，假设负载均衡后，每台节点上的DB数量应该为2或3，为了尽可能减少DB的迁移次数，则应该向新增节点上迁移2个DB，每台源节点只需迁移一个DB。首先，系统读取新增节点所在的交换机下原有节点中存储的DB信息，即数据块D1、G2、J3，确保不会再有相同的数据块副本D、G和J迁移过来；然后以11号节点为目的节点，分别从1号节点向后轮循，查找合适的DB进行迁移。首先在1号节点和11号节点之间，选择了A1作为迁移对象，由于1号节点已经达到均衡状态，继续轮循选择2号节点作为源节点并迁移数据块B1至节点11，由于此时节点11达到均衡状态，则选择节点12为目的节点，重新从1号节点轮循选择源节点。因为1号、2号节点都达到均衡状态，系统选择3号节点作为源节点并迁移数据块C1至节点12，然后在选择节点4作为源节点时，由于在10~12号节点所在的交换机下，已经存储有A1数据块，所以4号节点的数据块A2不能作为迁移对象，则选择了A2的下一个数据块D3并迁移至节点12。此时新增节点都达到均衡状态，即新增节点后对该字段的均衡操作完成，存储状态如表4-2所示。同理，分别对其他字段进行均衡操作，即可使得集群中数据存储重新达到基于字段均衡且各DB副本跨交换机存储。

表4-2 新增节点部署在原有交换机下的负载均衡后存储状态

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|  |  |  | A2 | B2 | C2 | A3 | B3 | C3 | D1 | A1 | C1 |
| D2 | E1 | F1 |  | E2 | F2 | E3 | G1 | F3 | G2 | B1 | D3 |
| G3 | H1 | I1 | H2 | I2 | J1 | H3 | I3 | J2 | J3 |  |  |

4.2.2新增节点部署在新交换机下

表4-3和4-4中分别表示了某一字段在集群新增节点之后的负载均衡效果。假设每台交换机下都部署三台节点，即1~3、4~6、7~9号节点分别部署在三台交换机下，其中由于原来每台交换机下节点均达到饱和状态，所以新增节点部署在新的交换机下。当集群新增一台交换机，且该交换机下部署了三台空闲节点，负载均衡前的存储状态如表4-3所示。

表4-3 新增节点部署在新交换机下的负载均衡前存储状态

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| A1 | B1 | C1 | A2 | B2 | C2 | A3 | B3 | C3 |  |  |  |
| D1 | E1 | F1 | D2 | E2 | F2 | D3 | E3 | F3 |  |  |  |
| G1 | H1 | I1 | G2 | H2 | I2 | G3 | H3 | I3 |  |  |  |

在该集群中，1~9号节点为原有节点，10~12号节点为新增节点。数据块总量为27，负载均衡后每台节点上的DB数量应为2或3。为了尽可能减少DB的迁移次数，则应该向新增节点上迁移2个DB，每台源节点只需迁移一个DB。在DB迁移之前，系统首先读取可知新增交换机下的所有节点DB存储信息为空，此时，可移动节点1中数据块A1到10号节点，此时节点1达到均衡状态。然后移动节点2的数据块B1至10号节点，节点2和节点10都达到均衡状态。依次系统在选取11号节点为目的节点，从1号节点开始轮循，由于1号和2号节点都处于均衡状态，则迁移节点3的数据块C1至节点11，在节点3达到均衡状态之后，继续选择节点4为源节点，由于10号节点已经存储数据块A1，节点4中数据块A2不能作为迁移对象，轮循选择节点4中的数据块D2并迁移至节点11。依次类推，直到节点6中的数据块F2迁移至节点12，该字段在扩容后的集群中达到存储均衡状态。

表4-4 新增节点部署在新交换机下的负载均衡后存储状态

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1号交换机 | | | 2号交换机 | | | 3号交换机 | | | 4号交换机 | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|  |  |  | A2 | B2 | C2 | A3 | B3 | C3 | A1 | C1 | E2 |
| D1 | E1 | F1 |  |  |  | D3 | E3 | F3 | B1 | D2 | F2 |
| G1 | H1 | I1 | G2 | H2 | I2 | G3 | H3 | I3 |  |  |  |

经过以上分析，初步推测不管新增节点部署在原有交换机还是新增交换机下，都不影响该负载均衡算法对集群中DB位置的调整，最后使得集群中每个字段基于字段均衡存储，且不同备份仍旧存储在不同的交换机下。

4.3小结

基于第三章的字段均衡的跨交换机分布式存储调度算法，本章继续提出了在集群中各字段DB副本已经达到均衡存储状态时，新增节点之后如何将各DB副本重新调整为均衡存储状态的负载均衡算法。由于仍要保证每个字段的DB在各节点均衡存储，该算法分别对每个字段进行均衡操作。只要使得每个字段在新增节点之后的集群中，达到均衡存储，整个结群的数据存储也就达到均衡状态。该算法每次选择迁移DB时，都对其进行判别是否新增节点所在交换机下的所有节点上是否已经存储，从而保证了各DB副本仍旧跨交换机存储。另外，通过仿真证明了不管新增节点部署在原有交换机下还是新增交换机下，都不影响该负载均衡算法的工作。

第五章 仿真结果与分析

5.1字段均衡的性能与分析

5.1.1字段均衡性

（1）单个字段的均衡存储仿真实验

从集群外客户端向集群上传10000个DB，每个DB存储三份，在整个存储调度过程管理节点将进行30000次计算，分配每个DB的存储位置。如图5-1所示，横坐标为节点序号，纵坐标为相应节点上分配存储的DB数目，更加直观地对比基于字段均衡的分布式存储调度算法和HDFS中分布式存储调度在数据存储均衡上的优劣性。

图5-1 20个节点规模集群中各节点DB个数分布图

经过对比发现，HDFS中分布式存储调度策略有一定的随机性，每台节点上的DB数目在平均范围内波动，而基于字段均衡的分布式存储调度策略则能够保证每台节点上DB数目近乎相等。平均范围内波动，而基于字段均衡的分布式存储调度策略则能够保证每台节点上DB数目近乎相等。

在任务计算时，假设集群中每台计算机各项性能指标相同，一台节点处理一个DB所用时间为一个单位时间，以图3分配存储结果为例，在一般分布式存储策略中，由于17号节点存储DB最少，将最先结束任务计算，相应地，作为存储DB最多的节点，2号节点计算任务持续时间最长，为515个时间单位，即为整个集群的任务计算时长。在基于字段均衡的分布式存储策略中，每台节点上DB存储数目几乎相同，则所有节点将同时完成各自的计算任务，所耗时长500个时间单位，即为整个集群的计算任务时长，比普通分布式存储节省了15个时间单位。所以，基于字段均衡的分布式存储调度策略能够将数据尽可能均衡地存储在各个节点上，使得集群中每台节点的计算任务量尽量均衡，从而使得整个集群的计算效率达到最高。

图5-2 25个节点规模集群中各节点DB个数分布图

图5-3 30个节点规模集群中各节点DB个数分布图

无论在哪种分布式存储方案中，系统存储文件的总时间都是由任务最后结束的那台节点决定的。当集群对某一字段数据进行任务调度时，假设集群中每台节点的性能相同，也就是说存储该字段数据最多的节点的任务执行时间即为集群对该字段的任务调度时间。如图5-1、5-2、5-3所示，20、25、和30台节点的模拟集群按照HDFS分布式存储策略进行测试，其中第2、4、和9号节点DB存储最多，在集群对该字段进行任务调度时，它们的任务负载也最大。同时，在三组测试中，17、12、和3号节点上存储该字段DB数最少，所以在整个集群任务调度结束之前，它们的空闲时间最长，造成了计算资源的浪费。

图5-1、5-2、5-3的测试结果表明，不管节点数量多少，基于字段均衡的分布式存储调度算法总能使得字段在各节点之间均衡存储。在HDFS分布式存储结果中，随着节点数量增加，发生个别节点存储DB数量远高于或远低于平均值的概率越大。尤其是当某一台节点上DB数量远高于其他节点上的DB数量时，不仅造成了部分节点长时间空闲，更重要的是拖长了整个集群的任务执行时间，影响了系统效率和用户体验。

（2）多字段均衡存储实验

一般情况下，数据库总要存储大量不同字段的数据。基于字段均衡的分布式存户算法对于多个字段的存储过程，其实就是分别对每个字段均衡存储。

如图5-4，在18节点的集群中，以存储A、B、C三中字段数据为例，且这三种字段数量分别为300、250、200，每个DB存储三份副本，即每个字段需要存储DB数量为900、750、600。其中A字段DB在每台节点上存储数量均为50，B字段DB在每台节点上的存储数量为41~42，C字段DB在各节点上存储数量为33~34，基本达到均衡。由此可见，对于一个字段而言，在集群中按照基于字段均衡的分布式存储策略进行存储，各台节点上的DB数量最多相差1。

图5-5展示了该集群中每台节点上总DB数量的分布，为124~126之间。因为每个字段在各节点上存储DB数最多相差1个，所以A、B、C三个字段存储完成之后各节点上的DB数量最多相差3个。由于A字段在各节点上存储恰好完全均衡，所以图5-5中各节点上DB数量极差为2。

图5-4 三种字段在各节点上存储DB数量

图5-5 各节点上DB总量分布

由此推测，在某种极端情况下，当客户端上传了大量不同字段的数据，且每种字段DB数量较少，就会引起前后节点的存储总量不均衡的问题。假如客户端向该集群上传500个不同字段的存储请求，每个字段的DB数量随机在1~5范围内，且都存储三份副本，按照基于字段均衡的分布式存储策略进行存储，各节点的数据存储状态如图5-6所示。

图5-6 大量不同字段在算法优化前的存储结果

由于节点数量正好是3的整数倍，所以DB副本正好按照严格轮循顺序在1~18号节点中进行存储。因为每个字段的DB数量最多为5，所以最多轮循至17号节点，即18号节点上存储DB数量为0。因为每个字段第一个DB的三份副本都从1号、4号和7号节点开始存储，第二份DB的三份副本分别存储在2号、5号和8号节点上，第三份DB的三份副本分别存储在3号、6号和9号节点上，第四份DB的三份副本分别存储在10号、13号和16号节点上，第五份DB的三份副本分别存储在11号、14号和15号节点上，由于每个字段的DB数量在1~5不等，所以该集群中DB存储状态如图5-6中所示，整体呈递减状态，各节点的DB存储数量分布很不均衡。

综上所述，在一般情况下，基于字段均衡的分布式存储策略不仅能够使各字段数据在节点之间均衡存储，而且还能使各节点上的存储总量基本达到均衡。但是当字段数量较多时，可能会引起节点上总的DB数量差别较大。

5.1.2大量不同字段存储优化

如图5-7，是对图5-6存储状态的优化结果。在优化后的基于字段均衡的分布式存储算法中，Master节点在选取DB存储位置时，除了比较每台节点上该字段DB的数量，还会比较选中节点上DB总数量与集群中最后一台节点上的DB总数量，在满足字段均衡的基础上，当它们的DB总数量相差不超过5时，才会被才选择为当前DB的存储节点。

图5-7 大量不同字段在算法优化后的存储结果

在存储过程中，由于采用从前向后的轮循方法，12号、15号和18号节点中DB数量大部分时间都滞后于其他节点。1号、4号、7号节点率先与18号节点相差5个DB，其次是2、5、8号节点。当其他节点都比18号节点多5个DB时，系统才会选择12号、15号和18号节点作为存储节点继续存储数据。

根据仿真结果可知，优化后的基于字段均衡的分布式存储调度算法不仅能够保证各字段数据在节点之间存储均衡，还能保证各节点上的存储总量达到基本均衡的状态。

5.2新增节点后负载均衡仿真

如图5-8，展示了新增节点部署在原有交换机下某一字段的负载均衡前后的数据存储状态。如图中负载均衡前折线所示，扩容前集群有17台节点，其中1~5、6~10、11~15、16~17号节点分别在不同的交换机下。在基于字段均衡的跨交换机存储中使得每台节点均衡存储了一种字段的数据，每台节点上有399~400个DB。当集群中新增三台节点至16~17号节点所在的交换机下时，集群规模扩大到20台节点，同时为了保证数据存储的均衡性，应该将原有的1~17号节点上部分DB迁移至18~20号节点，且保证每个DB的不同副本仍存储在不同的交换机下。

理论情况下，负载均衡完成后每台节点上应该存储的DB数量M约为：

即应该向17~20号新增节点上分别迁移340个DB。应该从集群中原有的1~16号节点移走的DB数量N约为：

由图5-8中负载均衡后折线可知，负载均衡后，每台节点上该字段的DB数量均在340左右，与理论分析相符。

图5-8 新增节点部署在原有交换机下的负载均衡效果

如图5-9，展示了新增节点部署在新交换机下某一字段的负载均衡前后的数据存储状态。集群中原有15台节点，其中1~5、6~10、11~15号节点分别部署在同一台交换机下，且假设每台交换机下最多部署五台节点。通过基于字段均衡的跨交换机分布式存储策略向原集群中存储2000个某一字段的DB，每个DB存储三份副本，则整个集群一共存储6000个DB，平均每台节点存储400个。现对集群进行扩容，新增5台节点，由于原集群中三台交换机下节点数量都已达到饱和，只能为新增节点重新部署一台交换机。

理论情况下，负载均衡后每台节点的DB存储数量应该为：

即应该向16~20号新增节点上分别迁移300个DB。应该从集群中原有的1~15号节点移走的DB数量N约为：

由图5-9中负载均衡后折线可知，负载均衡完成后，集群中每台节点上的DB数量均为300个，同时证明了不管新增节点部署在原有交换机下还是新的交换机下，该负载均衡算法都能完成数据的均衡操作。

图5-9 新增节点部署在新增交换机下的负载均衡效果

5.3小结

本章主要围绕基于字段均衡的分布式存储算法仿真实现展开，不仅验证了大部分情况下该算法能够使得数据在集群中基于字段均衡存储，还证明了优化后的算法能够解决大量不同字段在存储时引起的节点存储量不均的问题，在字段和数据总量上双重保证了节点之间的均衡存储。

另外，在基于字段均衡的分布式存储策略实验结果上，本章实现了集群在新增节点之后的负载均衡仿真实验，证明了第四章提出的负载均衡算法可以使得集群在增加节点之后的存储不均状态重新调整均衡，使每个字段在各节点上的存储DB数量差缩小到1。

第六章 结论与展望

6.1结论

本文通过对列式数据库中字段数据在分布式系统中的存储进行研究，基于HDFS分布式存储调度算法中存在的字段数据在节点之间存储不均衡的问题，提出了一种基于字段均衡的分布式存储调度算法。该算法在对列数数据存储时，对每个字段的数据分别对待，使每个字段在各节点上的存储都达到均衡状态，同时保证了数据库中各个节点上的存储总量基本相当。之后，对该算法进行了优化，解决了大量不同字段存储可能引起的堆积问题。另外，文中还提出了集群新增节点之后的负载均衡策略，在保证副本跨交换机存储的原则上重新调整集群数据存储，使每个字段的数据在各节点上再次达到均衡存储的状态。

本课题主要围绕分布式数据库存储的均衡性进而可靠性问题，主要工作和研究结论如下：

（1）对HDFS中分布式存储调度算法进行研究。针对其在列式数据存储时不能保证字段在节点时间均衡的问题，提出了一种基于字段均衡的跨交换机分布式存储调度算法。如第五章实验结果所示，该算法不仅保证了集群中各节点上存储数据总量相当，还保证了每个字段数据在各节点上的均衡存储，且均衡效果较好，不同节点之间相同字段的DB数量相差不超过一个。

（2）对基于字段均衡的分布式存储策略进行仿真实验。在该算法中，同一DB的不同副本是跨交换机存储。一般情况下对数据进行三副本存储时，在HDFS中，三份副本分别存储在两台交换机下，所以集群中最多容忍一台交换机故障；在基于字段均衡的跨交换机分布式存储调度算法中，三份副本被存放在三台交换机下，在集群中任意两台交换机故障的情况下，系统仍然能够继续工作。所以，在三副本存储原则上，基于字段均衡的跨交换机存储调度算法的可靠性是HDFS中分布式存储调度算法两倍。

（3）模拟集群故障时的数据恢复。副本跨交换机的存储方案虽然在一定程度上提高了集群的可靠性，但是集群故障是不可能避免的。文中分别对集群中单个DB故障、节点故障和交换机故障分别进行研究，针对这些不同的故障，提出了数据修复的方案；另外，文中还模拟了故障修复之后多余副本的删除工作，证明了该方案的可行性。

（4）对集群新增节点后的负载均衡进行仿真实验。随着存储数据量的增大，集群有时需要增加节点以扩大存储容量。当集群在新增节点时，原本各字段基于节点均衡存储的状态被打破，而且为了及时利用新增的节点资源，负载均衡算法将原节点中的部分数据迁移到新增节点上。根据第五章的仿真实验，证明了在保证副本跨交换机存储的原则上，该负载均衡算法能够使得新增节点时候的集群存储尽快达到新的均衡状态，且均衡效果与基于字段均衡的分布式存储调度算法一致。

6.2不足与展望

本文对分布式存储调度的均衡性和可靠性进行了研究分析，不仅优化了分布式数据库在存储列式数据的均衡性能，还提高了数据库中各DB副本存储的可靠性。但是由于资源和时间有限，仍存在一些不足之处，有待于更多的研究和改进。今后需要完善的工作主要有以下几个方面：

（1）在分布式数据库搭建的时候，为了尽可能简化集群结构，常常采用网口较多的交换机，因为这种交换机下可以部署的节点较多。但是为了提高数据存储的可靠性，文中提出的基于字段均衡的分布式存储调度策略中同一DB的不同副本时跨交换机存储的。所以即使集群中节点数量较少时，也需要交换机的数量不少于副本数量。

（2）虽然集群节点之间网络一般通过有线连接，但是由于数据传输量较大，所以带宽资源比较宝贵。文中基于字段均衡的分布式存储调度算法和负载均衡算法都保证了字段在各节点上的均衡存储、副本跨交换机存储，但都没有考虑集群中数据传输时的带宽。后续的工作中应该在保证均衡性和可靠性的基础上，在数据进行传输时应选择“距离较近”节点进行传输，尽量提高文件传输效率。

（3）本文中的存储策略都是以最常用的每个DB存储三份副本设计的，还没有将备份数量的选择向用户提供接口。如果入户可以根据数据的重要性动态地选择备份级别，将在很大程度上提高数据库的可用性。

（4）在对大量不同字段存储可能导致堆积问题的优化方案中，每个DB在存储时都比较当前节点中DB数量与集群中最后一台节点上DB数量，在它们相差超过5个DB时，继续轮循下一个节点。同样，这里相差的DB数量也没有向用户提供接口，在以后的改进工作中，可以提供给客户端选择的权限，根据集群中节点的性能做出灵活选择。

参考文献

1. Bhowmick S, Chakraborty S, Agrawal D P. Study of Hadoop-MapReduce on Google N-Gram Datasets[C]. IEEE, International Conference on Mobile Ad Hoc and Sensor Systems. IEEE, 2016:488-490.
2. 阳振坤, 杨传辉, 李震. 海量结构化数据存储管理系统OceanBase[J]. 科研信息化技术与应用, 2013, 4(1):41-48.
3. 杨传辉. 大规模分布式存储系统:原理解析与架构实战[M]. 机械工业出版社, 2013.
4. Dong B, Zheng Q, Tian F, et al. An optimized approach for storing and accessing small files on cloud storage[J]. Journal of Network & Computer Applications, 2012, 35(6):1847-1862.
5. Abadi D, Madden S, Ferreira M. Integrating compression and execution in column-oriented database systems[C]// ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, Usa, June. DBLP, 2006:671-682.
6. Abadi D J, Boncz P A, Harizopoulos S. Column oriented Database Systems.[J]. Proceedings of the Vldb Endowment, 2009, 2(2):1664-1665.
7. Abadi D J. Query execution in column-oriented database systems[J]. Massachusetts Institute of Technology, 2009, 2(2):671--682.
8. Yan W, Li C L, Du S, et al. An Optimization Algorithm for Heterogeneous Hadoop Clusters Based on Dynamic Load Balancing[C]// International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE, 2017:250-255.
9. Liu Y, Jing W, Liu Y, et al. A sliding window‐based dynamic load balancing for heterogeneous Hadoop clusters[J]. Concurrency & Computation Practice & Experience, 2017, 29(3):n/a-n/a.
10. Kumar S, Singhal N. A Study on the Assessment of Load Balancing Algorithms in a Grid Based Network[J]. International Journal of Soft Computing & Engineering, 2012, 2(1).
11. Li W, Zhou J, Lin Z, et al. Hadoop-Based Dynamic Load Balance Scheduling Algorithm of Logistics Inventory[C]// International Conference on Computational Intelligence and Security. IEEE, 2017:5-8.
12. Yadav S, Vishwakarma S, Verma A. Efficient & Accurate Scheduling Algorithm for Cloudera Hadoop[C]// International Conference on Computational Intelligence and Communication Networks. IEEE, 2016:839-844.
13. Hou X, Pal D, Ashwin K T K, et al. Privacy Preserving Rack-Based Dynamic Workload Balancing for Hadoop MapReduce[C]// IEEE, International Conference on Big Data Security on Cloud. IEEE, 2016:30-35.
14. 黄建忠,曹强,黄思倜,谢长生. 面向纠删码存储集群的节点并发重构[J]. 计算机研究与发展,2016,53(09):1918-1929.
15. 章宏灿,薛巍. 集群RAID5存储系统可靠性分析[J]. 计算机研究与发展,2010,47(04):727-735.
16. Ghemawat S, Gobioff H, Leung S T. The Google file system[C]// Nineteenth ACM Symposium on Operating Systems Principles. ACM, 2003:29-43.
17. Jung D, Lee K, Nam Y, et al. Load Balancing for Improving Hadoop MapReduce Performance in Cloud-based Cluster Environment[J]. 한국정보과학회 학술발표논문집, 2013.
18. Liu Q, Cai W, Shen J, et al. VPCH: A Consistent Hashing Algorithm for Better Load Balancing in a Hadoop Environment[C]// Third International Conference on Advanced Cloud and Big Data. IEEE, 2015:69-72.
19. Srinivasan L, Varma V. Adaptive Load-Balancing for Consistent Hashing in Heterogeneous Clusters[C]// Ieee/acm International Symposium on Cluster, Cloud and Grid Computing. IEEE, 2015:1135-1138.
20. Lin C Y, Lin Y C. A Load-Balancing Algorithm for Hadoop Distributed File System[C]// International Conference on Network-Based Information Systems. IEEE, 2015:173-179.
21. Li H, Hou C, Wang J. Reliability analysis of Hadoop cluster System based on proportional hazards model[C]// IEEE International Conference on Software Engineering and Service Science. IEEE, 2017:132-138.
22. Wang K, Yang Y, Qiu X, et al. MOSM: An approach for efficient storing massive small files on Hadoop[C]// IEEE, International Conference on Big Data Analysis. IEEE, 2017:397-401.
23. Zhang Y, Liu D. Improving the Efficiency of Storing for Small Files in HDFS[C]// International Conference on Computer Science and Service System. IEEE Computer Society, 2012:2239-2242.
24. Xu W, Lu Y. Energy Analysis of Hadoop Cluster Failure Recovery[C]// International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE, 2013:141-146.
25. Das S K, Roy S, Sarker D, et al. Energy Efficient Failure Recovery with Concurrent Execution in Hadoop Cluster[J]. 2016, 7(1):8.
26. 熊润群. 面向大数据的异构集群存储系统副本放置与选择策略研究[D].东南大学,2015.
27. 罗鹏, 龚勋. HDFS数据存放策略的研究与改进[J]. 计算机工程与设计, 2014, 35(4):1127-1131.
28. Li M, Li R, Lee P P C. Relieving Both Storage and Recovery Burdens in Big Data Clusters with R-STAIR Codes[J]. IEEE Internet Computing, 2016, PP(99):1-1.
29. Luo S, Wang Y, Huang W, et al. Backup and Disaster Recovery System for HDFS[C]// International Conference on Information Science and Security. IEEE, 2017:1-4.
30. 郑惠瀚. 基于HDFS的分布式存储中负载均衡技术研究[D]. 哈尔滨工业大学, 2016.
31. Wang Y, Huang W, Luo S, et al. Fast off-site backup and recovery system for HDFS[C]// The, International Conference. 2016:19-23.
32. Whitehead J, Montulli L, Harrison J, et al. Systems and methods for on-line backup and disaster recovery[J]. 2016.
33. Zheng W. System for disaster recovery backup and a method thereof[J]. 2012.
34. Qin Y, Ai X, Chen L, et al. Data placement strategy in data center distributed storage systems[C]// IEEE International Conference on Communication Systems. IEEE, 2017:1-6.
35. Liu X. A Strategy of Data Placement in Distributed Storage System[J]. Computer & Digital Engineering, 2009.
36. Lian Q, Chen W, Zhang Z. On the Impact of Replica Placement to the Reliability of Distributed Brick Storage Systems[C]// IEEE International Conference on Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. IEEE, 2005:187-196.
37. Wu J, Hong B. Multicast-based replication for Hadoop HDFS[C]// Ieee/acis International Conference on Software Engineering, Artificial Intelligence, NETWORKING and Parallel/distributed Computing. IEEE, 2015:1-6.
38. Ghemawat S, Gobioff H, Leung S. File and storage systems: The Google File System[J]. Acm Symposium on Operating Systems Principles Bolton Landing, 2003, 37:29-43.
39. 薛建生, 于忠臣, 黄磊,等. 物联网海量数据的分布式存储算法[J]. 小型微型计算机系统, 2013, 34(9):2081-2084.

致谢

时间过得好快，两年半的研究生生活转眼就要过去了。想起从刚踏入实验室的自己，跟着师兄师姐们学习看论文、做项目，甚至有了生活上的烦恼也去向他们求助，这里发生的每件事、接触的每个人都让我充满了感激之情。这两年半的时间里，我收获了太多东西，也成长了很多。从第一次参与做项目，再到阅读大量文献、做研究、写仿真，直到最后坚持到毕业论文的完成，虽然付出了很多艰辛与汗水，但我最后感受更多的还是充实与幸福。经过这两年半的学习，我的专业能力和各方面的素养都得到了很大的提升，我为自己能来到这个实验室而感到自豪！

首先，我要感谢我的导师别红霞教授。在学术方面，别老师根据我个人的兴趣与基础，为我精心挑选了数据存储的毕设方向，在我遇到困难时，也总能根据她丰富的学术经验为我耐心讲解分析，在学术研究过程中不断为我指明方向。同时，别老师严谨的学术态度也让我收获颇丰，在论文未开题时就先锻炼我们的写作能力，使我的论文表述和写作能力有了很大提升；另外别老师也在我们的日常生活中给了很大的鼓舞，不管是在实验室的谈话还是校园里不经意的聊天，处处都鼓励着大家，给大家培养了一种在快乐中科研的心态。总之，我在研究生期间的一切进步与成长都离不开别老师的悉心栽培，我也将终生感激别老师为我所做的一切。

其次，我还要感谢于泓老师。于老师是别老师为我请的一位指导老师，他凭借丰富的工程经验总是能在第一时间为我答疑解惑。尤其是在毕业设计过程中，每周给我开会指导，不厌其烦地为我讲解其中的各种相关理论。不仅如此，于老师还传授给我许多实践经验，为我以后的工作提供指导性意见。这些帮助也会使我受益终生。

最后，我还要感谢实验室的同学们。他们不仅在生活中是我的好朋友，在学习生活中也提供了很多帮助，不管是项目中的困难还毕业设计中的疑惑，他们总能及时地向我伸出援助之手，让我明白大家一起学习一起进步也是一件很幸福的事。

攻读学位期间取得的研究成果

[1] 李开放，别红霞 跨交换机的分布式存储调度机制 中国科技论文在线，http://www.paper.edu.cn/releasepaper/content/201712-35