

分类号_____

学校代码 10487

学号 M201676174

密级_____

华中科技大学

硕士学位论文

HDFS 管理系统的设计与实现

学位申请人：许田龙

学 科 专 业：软件工程

指 导 教 师：吴 涛 教授

答 辩 日 期：2018.12.25

**A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering**

Design and Implementation of HDFS Management System

Candidate : Xu Tianlong

Major : Software Engineering

Supervisor: Prof. Wu Tao

Huazhong University of Science and Technology

Wuhan 430074, P. R. China

December, 2018

独创性声明

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，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：汗田龙

日期：2019年 1 月 5 日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：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电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本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和汇编本学位论文。

本论文属于 ☐ 保密， ☐ 在_____年解密后适用本授权书。
☐ 不保密 ☒。

（请在以上方框内打“√”）

学位论文作者签名：

日期：2019年 1 月 5 日

指导教师签名：

日期：2019年 1 月 5 日

摘要

随着移动互联网以及物联网的迅速发展，其产生的数据呈指数级增长，传统技术对大量数据的存储及计算显得力不从心，Hadoop 应运而生，其分布式文件存储系统（HDFS）被广泛应用于的海量数据的存储。虽然 HDFS 解决了海量数据的存储问题，但是它也面临新的挑战。大数据的存储被期望有更多的场景，例如：运维更加高效智能；文件大小各异，HDFS 应该对大小文件都有很好的支持；数据冷热程度的不同，也应根据冷热程度的差异，存放在不同的介质上；应该具有更好的容灾备份等等。

为了解决上述问题，本系统针对 HDFS 的缺点对其进行改进，将集群的元数据以及访问记录存储到关系型数据库中，利用这些数据完成以下几个系统目标：冷热数据智能调度和迁移，HDFS 对小文件的支持和优化，更好的容灾备份策略，以及更易用的系统操作指令。

本系统采用 B/S 架构，使用 Java 作为开发语言，采用 Zeppelin 做为前端展示的框架，后台使用 Jersey 作为 RESTful Webservice 框架，MySQL 作为数据库。利用 Hazelcast 组件来抗单节点故障。用户可以通过浏览器界面输入运维指令，查看系统任务的运行状况。

系统按照标准的软件开发流程，有需求分析，系统设计，系统实现和系统测试四个阶段。开发完成后，经测试表明，系统可以稳定运行，没有明显错误，实现了设计目标，对 HDFS 的改进是行之有效的，大大改善了 HDFS 的使用场景及运维成本。

关键词：合并小文件 冷热数据 容灾备份

Abstract

With the rapid development of mobile Internet and Internet of Things (IOT), the data generated by the mobile Internet and IOT grows exponentially. Traditional technology can not store and calculate a large amount of data. Hadoop emerges as the times require. Hadoop Distributed File System (HDFS) is widely used in the storage of massive data. Although HDFS can solve the storage problem of massive data, it is facing new challenges. Big data storage is expected to have more scenarios. For example, operation and maintenance should be efficient; HDFS should have good support for small files; data should also be stored in different media according to the degree of heat and cold. It should have better capacity of recovery and backup and so on.

In order to solve these problems, the system synchronizes metadata and access records of cluster to relational database. Using these data, the system refines HDFS's shortcomings and achieve the following goals: intelligent migration of hot data and cold data, optimization of small file, better recovery and backup, and easier-to-use system operation instructions. The system uses B/S architecture and chooses Java as the development language, Zeppelin as the Web display framework, Jersey as the RESTful Web Service framework, MySQL as the database. Hazelcast component is used to resist single node failure. Users can input commands through the browser and check the status of the tasks.

According to the standard software development process, the system has four stages: requirement analysis, system design, system implementation and system testing. After the development, the test shows that the system can run stably without obvious errors, and achieves the design goal. The improvement of HDFS is effective, and it greatly improves the function and operation of HDFS.

Keywords: Merge small files Hot data and cold data Disaster recovery

华中科技大学硕士学位论文

目 录

摘 要	I
Abstract	II
1 绪论	1
1.1 研究背景与意义.....	1
1.2 国内外研究概况.....	2
1.3 论文主要研究内容	4
2 系统需求分析	5
2.1 系统功能需求.....	5
2.2 非功能性需求.....	7
2.3 本章小结.....	7
3 系统设计	8
3.1 系统架构设计.....	8
3.2 主要功能模块设计	9
3.3 系统数据库设计.....	23
3.4 本章小结.....	29
4 系统实现	30
4.1 开发环境与工具.....	30
4.2 系统部署与配置.....	31
4.3 系统主要功能实现	33

华中科技大学硕士学位论文

4.4 本章小结.....	42
5 系统测试	43
5.1 系统测试环境.....	43
5.2 系统功能测试.....	44
5.3 非功能性测试.....	47
5.4 本章小结.....	49
6 总结与展望	50
6.1 总结.....	50
6.2 展望.....	50
致 谢.....	52
参考文献.....	53

1 绪论

1.1 研究背景与意义

随着移动互联网和物联网的发展，人们产生的数据迅速增长，随之产生的大数据技术层出不穷，Hadoop 是最重要也是最基础的大数据存储和分析平台，其分布式的存储和计算策略，使得其扩展性，容错性都非常优异，用户可以以较低的成本在 Hadoop 集群上存储和分析海量数据^[1,2]。

HDFS 是其最为核心的一部分，为用户提供分布式文件存储的功能，因其支持水平扩展以及备份功能，被各大公司及其科研院所广泛采用^[3]。大数据分析框架虽然日新月异，出现了比如 Hive ,Spark,KyLin 等优秀的框架^[4]，但是其底层的存储系统仍然采用 HDFS,所以 HDFS 在大数据生态中占有举足轻重的地位。但是 HDFS 是 Hadoop 原生的文件系统，存储场景支持并不丰富，管理也不灵活，运维难度比较大，如何对 HDFS 进行改进，开发一个管理 HDFS 的系统^[5,6]，使其有更多的使用场景具有深远的意义。

本文研究意义主要有以下几点：

1) 使用本系统，可以轻松实现对 HDFS 的管理。原生的 Hadoop shell 不能做或者很难做的操作，利用本系统可以轻松实现。比如：每天定时删除 HDFS 文件系统中存在时间超过 2 个月，并且以 log 为结尾的文件。这种操作原生的 shell 命令并不容易做，但是利用本系统可以一条指令就可以做到。不仅如此，系统更提供了任务的展示界面，指令产生的结果在 Web 页面会有直观的展现，任务成功或者失败都能一目了然，任务的失败信息也会在前端页面展示出来，这样就大大降低运维难度。

2) HDFS 中文件的冷热并不相同，热文件是指被访问，改变较多的文件，冷文件与之相反。冷热程度不同的文件，存储策略也应该不同。比较热的数据，可以将文件数据放到内存中或者 SSD 存储介质上，来加快 IO 速度^[7,8]，也支持只将数据的某一个备份放到 SSD 上，普通的文件数据可以放到一般的存储介质上，还有些需要归档的冷数据，可以将其放到存储介质更差，计算性能也差的机器上面。这种灵活的

存储策略，既可以保证热数据较高的 IO，加快数据处理速度，又能使大量的冷文件使用廉价的机器，来节约资源，效益最大化。本系统支持数据的不同存储策略，用户可以自己指定文件的存储策略，系统会帮用户完成文件块的移动，这样来实现存储的最大效益。

3) HDFS 本身就是为存储大文件而生，它将大文件分成一个一个的块，达到分布式存储大文件或者超大文件的目的，但是 HDFS 对于小文件的支持并不友好，如果小文件过多，那么海量小文件的元数据将占据集群 NameNode 的大部分内存，甚至会导致机器崩溃，这样就降低了系统的稳定性和速度。Hadoop 的分布式运算框架 MapReduce^[9]，会根据切片来开启 Map 任务，如果是小文件，本身至少会占一个切片，从而启动一个 Map 任务。而任务启动将耗费大量时间甚至大部分时间都耗费在启动任务和释放任务上，所以小文件的优化是非常有必要的。本系统针对 HDFS 这一弊端，设计并完成了小文件的合并策略，使得用户可以非常方便地合并小文件。

4) HDFS 本身就是高可用的文件系统^[10]，不是非常极端的情况，不会产生数据丢失或者系统崩溃的状况。但是可能由于运维人员的误操作，或者黑客的攻击，导致集群数据丢失，或者系统奔溃，这会给企业造成灾难性的损失。本系统针对这种情况，系统支持 Kerberos 进行权限管理，而且实现了集群备份。使用本系统可以轻松实现集群间的同步或者异步备份，从而进行数据恢复。

1.2 国内外研究概况

回顾 Hadoop 的发展，国外公司占据指导地位，Hadoop 本身就起源于国外公司 Google，它是云计算概念的最先提出者，凭借本身在搜索业务中大量文件的存储积累，创造性的提出 GFS^[11]，从此文件存储进入分布式时代。Google 公司发表 GFS、MapReduce 以及 BigTable 三篇云计算领域核心技术论文，开源社区的开发人员对其论文进行了开源实现，后来 Apache 基金会整合以前的开发人员的贡献，开发推出了 Hadoop 生态系统，自此 Hadoop 生态发展迅速，已经成为业界大数据存储和分析的主要工具，在大数据生态中占有最重要的地位。

1) 在国内，大部分公司还是使用原生的 HDFS 文件系统作为存储组件，针对

HDFS 的一些弊端，比如海量小文件问题，国内的有很多的解决方案^[12-14]，比如 WebGIS，这是结合 web 和地理信息系统(GIS)而诞生的一种新系统^[15]。论文的主要设计思想是将小文件合并成一个大文件，并且保留小文件在大文件中的索引信息，但是这都是针对 HDFS 某一个问题产生的解决方案，基本没有一个系统的解决方案，各大公司也是针对 HDFS 进行一些调优工作，没有专门为 HDFS 建立一个管理系统。虽然有一些项目来使用 Web 页面来对 HDFS 进行管理，但是其系统非常的简单，只是一些简单的文件上传或者删除之类的简单操作，并没有针对 HDFS 的弊端来进行改进，所以国内对于 HDFS 改进的贡献是非常有限的。

随着大数据技术的发展，以及国内公司技术实力的增强，如何使得 HDFS 适应其公司业务的发展成为迫切需求，其必将间接导致国内加大对 HDFS 系统的管理和改进，国外在大数据技术方面的优势是我们前进最大的动力。

2) 在国外，因为其 Hadoop 本身就是在美国发展起来的，基本上 HDFS 所有的改进以及解决方案都由国外提供，由 Hadoop 以及大数据的生态发展来看，国外占绝对的领导地位。在 HDFS 的进化上，有很多公司对其有卓越的贡献。Federation，HA 等等改进离不开国外 IT 公司 FaceBook 等的技术支持。很多 HDFS 的弊端，随着 HDFS 的发展也基本上可以得到解决^[16-18]，比如关于集群数据恢复中，HDFS 其实也有默认的实现方案 DistCp，但是这种方案虽然可以一定程度上解决数据恢复问题，但是使用时有局限性，比如 DistCp 不是一个实时备份，而且严重依赖于 MapReduce，性能开销很大。国外的小文件的解决方案中，Hadoop Archive 会利用归档工具将小文件合并成一个 HAR 文件，这种文件不可改变，有其局限性，改进这些局限性，这也是本系统存在的意义。

虽然国外在大数据技术方面优势明显，但是更多的是因为其大数据技术起步较早，而且又有 Apache 基金会的支持，所以在开源社区的发展以及成果远胜于国内，但是随着国内公司以及研究单位对技术的重视，以及开源社区在中国的发展，国内的大数据技术成绩斐然，阿里巴巴公司的分布式文件存储系统发展迅速，国内的 Kylin^[19]项目也成为了 Apache 顶级项目，相信不久的将来，国内大数据技术会继续进步，能够赶超国外。

1.3 论文主要研究内容

设计并实现一款 HDFS 的管理系统，是本论文的主要研究内容。期望本系统能够方便 HDFS 的运维工作，增加 HDFS 的应用场景，增强 HDFS 的安全性以及数据恢复的能力^[20-22]。系统采用 B/S 架构，根据具体的需求模块，按照标准软件开发流程，采用 Java 语言进行功能研发和测试，完成整个系统的开发工作。文章主要研究内容如下：

针对 HDFS 在业界的使用现状进行调研分析，分析开发此系统的必要性。对采用的开发语言以及开发框架做选择并简单介绍。

针对 HDFS 运维痛点以及 HDFS 本身的缺陷做出详细的需求分析，不仅有功能性需求，对于非功能性需求也有明确要求。

针对系统的功能性需求以及非功能性需求进行系统设计，确定系统的主体架构，以及对每一个模块的详细设计，确定每个模块的设计目标，并描述模块达到的功能以及使用方法。

结合需求分析以及系统设计，选定相应的硬件以及软件进行具体的系统实现，严格按照软件开发流程，对每一个系统模块进行具体的开发工作。

对系统进行功能性以及非功能性测试，基本达到了设计目标，测试用例全部运行成功，完成了系统的开发工作。

总结了本系统的特点，以及不足之处，提出了对本系统更高的期望，以适应日益发展的大数据技术。

2 系统需求分析

本章主要对系统的功能性需求以及非功能性需求进行分析。

2.1 系统功能需求

作为一款 HDFS 的管理系统，我们希望它不仅仅能够提供一管理系统的基本功能，比如对文件的增删改查这种基本功能，它应该具备更多更智能的运维操作，以及其他存储场景的支持。经过与企业生产环境中的运维开发人员讨论分析，充分调研，最后与开发小组人员深入挖掘其生产过程中的 HDFS 痛点，总结的功能性需求如下：

1) 更加智能的运维指令。系统可以在 Web 界面输入指令。指令分为两种，一种是较为简单的指令，我们称之为 Action 指令，指令是具体的而且会立马执行，比如删除 HDFS 中文件名为/test.log 的文件，这种指令就是 Action 指令。还有一种称之为 Rule 的指令，这种指令可以使用户指定任务的运行时间，触发条件，以及做某种操作。比如每天凌晨向另一个集群备份文件大小大于 20M 的文件。用户可以利用这两种指令，轻松实现自己想执行的操作。本系统的指令不仅仅能够支持 HDFS 已有的应用场景，而且支持更加复杂的存储场景，主要有以下几个方面。

(1) 冷热文件的不同存储策略

该部分首先会对用户访问文件的信息做统计，因为冷热文件的判定很大程度上要根据文件被访问的次数。对于文件的存储策略，根据文件冷热程度来进行存储。比如，热数据可以放到性能较好，存储介质为 SSD 的机器上，甚至可以将文件缓存到内存中。对于热度一般的数据，就放到普通机器上即可。至于访问较少的数据，可以对其进行归档处理，将数据放到综合条件比较差的机器上。

用户可以指定某个文件的存储策略，系统来实现文件块的移动。

(2) 小文件合并

用户会自己可以指定文件容器（小文件合并后的文件），也可以使用系统默认的文件容器。将小文件合并之后，支持对原来小文件的读和删除，因为 HDFS 本身

不支持文件的修改，所以不提供小文件的修改。用户可以输入指令，合并一批大小不一，类型不同的文件，文件容器可以不限大小，支持持续向文件容器写入数据。

(3) 多集群备份以及支持 AWS S3

用户需要在去搭建一个集群来进行备份，或者使用较为成熟的 AWS S3 来进行文件的备份工作。关于数据备份，有同步备份和异步备份两种方式。同步备份是指在向集群中写入数据时，同时向备份集群中写入数据。两个集群文件都写入成功才表示文件写入成功。异步备份，是用户指定集群中的已经存在的某些文件，向备份集群进行备份。

2) 直观的任务结果展示以及任务管理。该部分显示指令执行的状态以及结果。对于所有的指令，都应该有任务的结果展示，包括是否成功，运行所用时间等。如果运行过程中失败，在 Web 界面也应该有错误的 log 信息。对于 Action 指令，只需要显示结果就可以。对于 Rule 指令，因其有定时功能，会展示其指令的状态，如暂停，运行中等状态，也应该提供指令状态改变的切换功能。比如让一个暂停的 Rule 任务重新执行。

根据以上功能需求，构建用例图。系统管理员的用例图如图 2-1 所示。

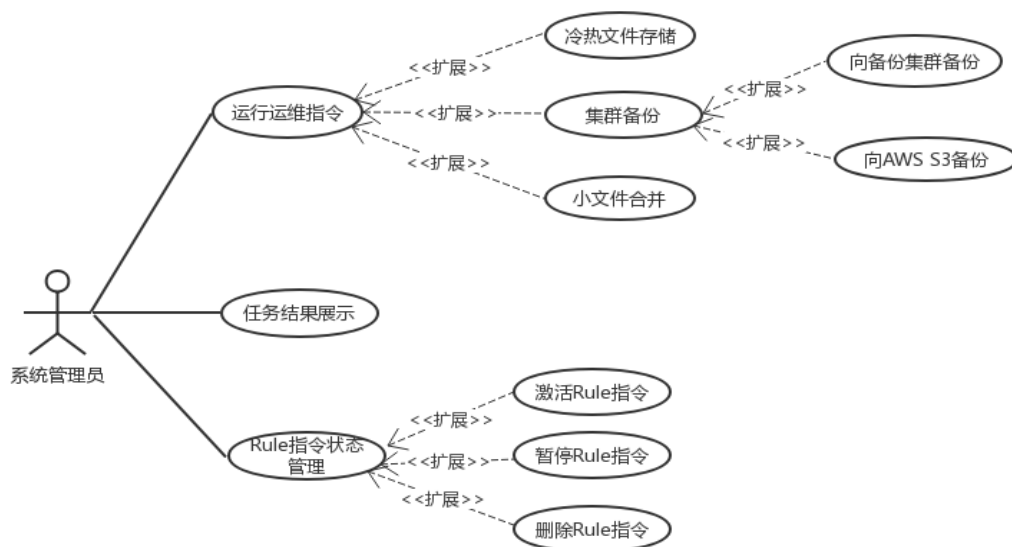


图 2-1 系统管理员用例图

2.2 系统非功能性需求

本系统在满足功能性需求之余，还应满足不同方面的非功能性需求。具体非功能性需求如下：

1) 易用性需求。系统本身就是为简化运维操作而诞生的，所以在易用性上要满足系统管理员方便地输入不同类型的指令。为 Rule 指令和 Action 指令开辟两块不同的区域提供指令输入。而且 Rule 指令要在界面上展示已经输入过的指令。对于指令运行的结果，成功的任务，应该显示为绿色，失败的任务应该显示为红色。任务执行结果按照时间进行排序，在单独一个页面进行展示。

2) 性能需求。正常的页面反应需求应在 0.5-2 秒之间，对于任务结果展示界面，因其展示的任务记录可能非常多，所以页面响应速度可以略微有延时。对于大量任务的提交，系统对任务的处理延迟不可低于 5 秒。并发量至少支持 500 个任务的同时运行。而且系统对 CPU 和内存的占用不可超过 90%。

3) 可靠性需求。当大量任务提交后，不会出现系统崩溃的状况。当系统宕机时，有备用机器来抗单节点故障，还要有数据恢复功能。允许出现少量错误，但不能影响系统其他任务的正常运行，打印错误的日志信息即可。

4) 安全性需求。支持 Kerberos^[24]认证，加强集群的安全性^[25-27]，只有系统管理员才可以对系统进行操作，而且对 HDFS 进行操作时会再进行身份验证，确保文件内容不会被不法份子破坏。

5) 兼容性。系统可以运行在 Linux 系统：CentOS 6.0 及 Debian 6.0 以上。HDFS 需是 2.7.3 及以后的版本。前端界面需兼容火狐，Chrome 以及 IE 浏览器。JDK 版本为 1.7 及以后版本。数据库为 MySQL 5.6 版本及以后版本。

2.3 本章小结

本章针对功能性需求以及非功能性需求进行了细致的研究和分析。确定了系统应该满足的各种需求，系统设计就应该以完成这些需求为目标，进行系统和各个模块的设计，以满足其功能性以及非功能性需求。

3 系统设计

3.1 系统架构设计

为完成系统的功能性需求以及非功能性需求，首先要对系统的整体架构进行设计。系统的层次架构设计如图 3-1 所示，不仅仅要对每层的要完成任务进行设计，还要对每层使用的技术和框架进行选择，使用合适的技术才能实现设计目标。下面是每层的详细说明。

Web 展示层：这一层是本系统的最上层，HDFS 系统运维人员就是通过 Web 层来输入指令，进而来达到管理 HDFS 的目的，本层还提供指令运行的结果展示以及状态管理，方便运维人员直观地了解任务运行状态。本层采用前端框架 Zeppelin，使用 RESTful^[28]接口与服务端进行通信。

服务层：本层是整个系统的核心，接受前端发送的请求，并返回结果。当用户在前端界面输入一条指令，会向服务端发送请求，服务端会解析指令，并向数据库查询所需要的 HDFS 元数据和其他数据，之后会执行指令。本系统是分布式系统，服务端有三种角色 Active Server, Standby Server 和 Agent Server。

Active Server：负责响应前端的请求，并将任务分发给 Agent Server 执行，减轻 Active Server 的压力。另一个重要任务是同步 NameNode 的元数据到数据库中，对用户访问 HDFS 的记录也要收集并存储到数据库中。

Standby Server：主要的作用是当 Active Server 宕机的时候，自动变为 Active Server，以此来抗单节点故障。另外当 Standby Server 是 Standby 状态时也会接收 Active Server 发送过来的任务并执行。

Agent Server：作用较为单一，就是分担 Active Server 的压力，执行 Active Server 分发过来的任务。

服务端使用 Akka 分布式软件开发工具^[29]，实现分布式架构，高并发异步执行任务。本系统使用 Hazelcast 集群管理工具实现抗单节点故障。

资源层：本层是系统的最底层，系统本身是在 HDFS 之上建立的一套管理系统，

所以很多任务的执行都要使用 HDFS 提供的接口。另外系统使用 MySQL 作为 NameNode 元数据的存储系统，每个任务的执行状态以及结果也会存在数据库中。

综上，系统的整体架构思路即服务端从 NameNode 中获取元数据以及用户访问 HDFS 的时间记录，存储到 MySQL 数据库中，接受前端页面的指令，利用在 MySQL 中的数据生成具体的对 HDFS 的操作并执行。

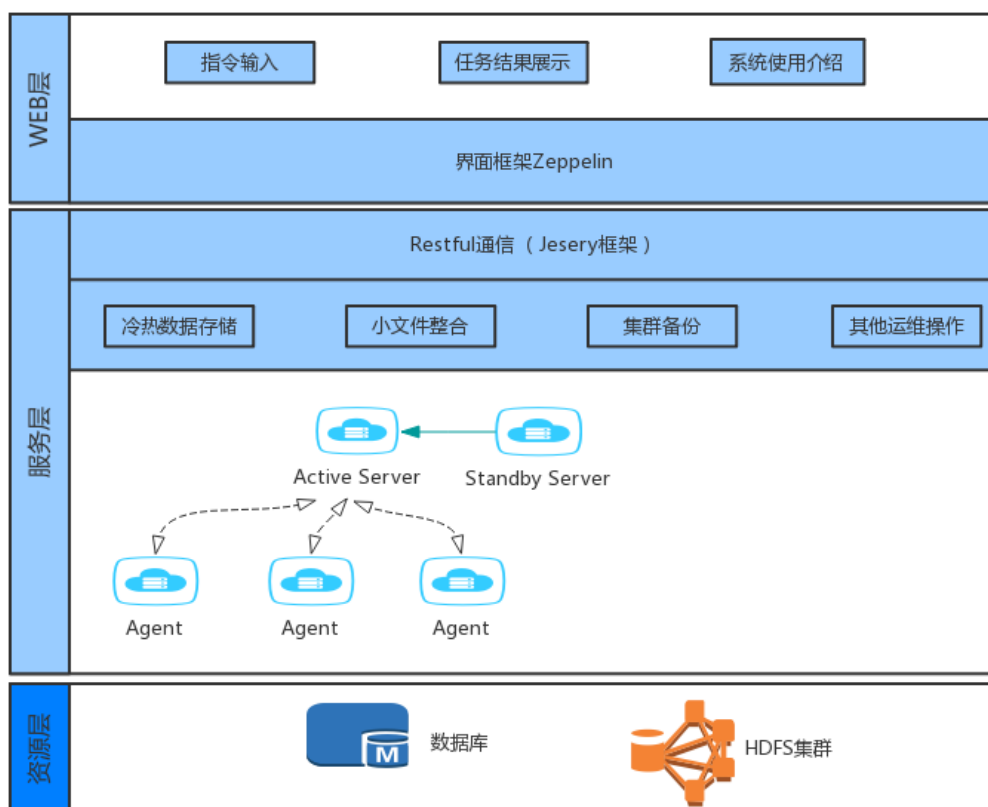


图 3-1 系统层次架构图

3.2 主要功能模块设计

经过认真的需求调研，针对功能性需求，对系统功能进行设计。主要分为冷热文件存储功能，小文件整合功能，集群备份功能，运维指令以及任务运行状态和结果查询。系统的功能模块如图 3-2 所示。

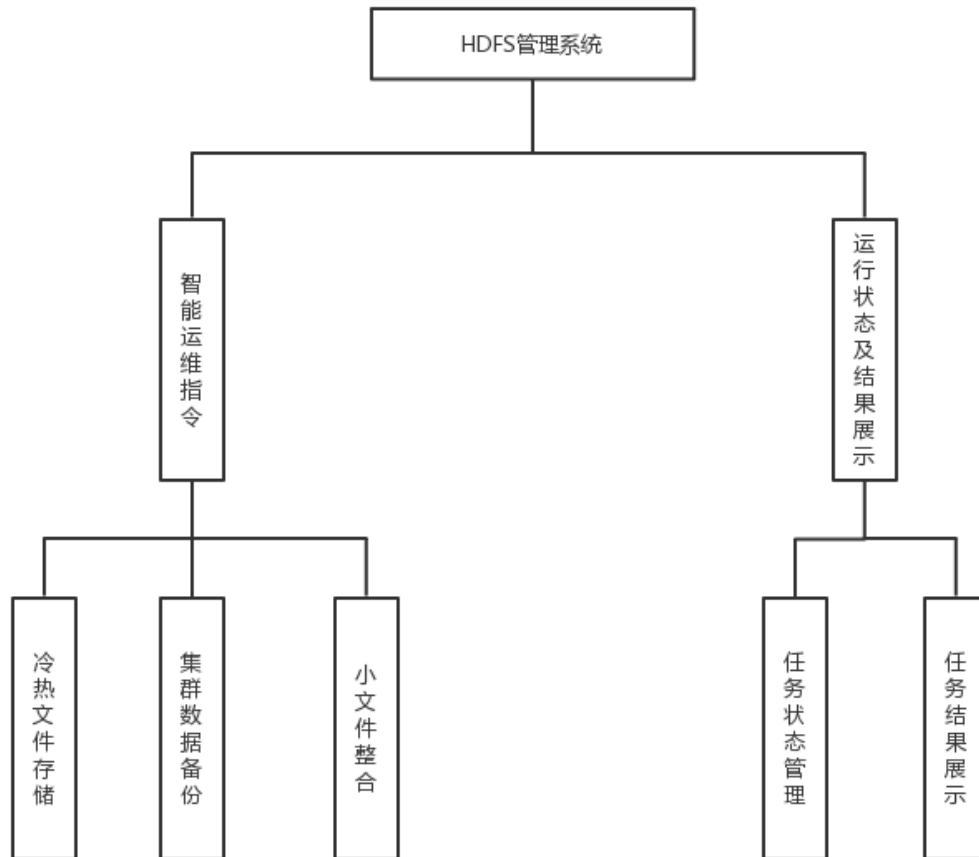


图 3-2 系统功能模块图

本章对某几个重要模块进行详细的设计描述，包括功能概述，流程设计以及数据流向和关键功能设计方案等。

3.2.1 运维指令设计

运维指令分为两种，一种是 Action 指令，一种是较为复杂的 Rule 指令，下面对这两种指令进行详细设计描述。

对于所有的指令，都会经过前端发送给 Active Server 进行解析，当执行的时候并不一定在 Active Server 这台机器上执行，因为可能由于任务太多，导致 Active Server 崩溃，本系统是分布式的系统，所以在具体执行的时候，它会将任务进行分

发给不同的 Agent Server，来减轻 Active Server 的压力，提高可靠性。系统运行过程可见图 3-3。

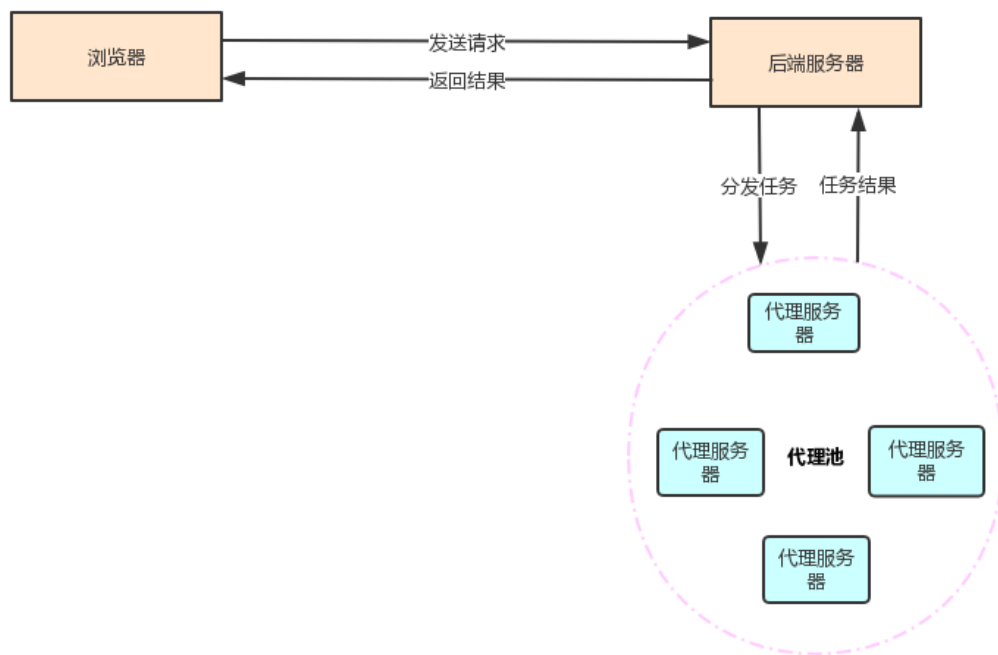


图 3-3 系统运行图

1) Action 指令设计

Action 指令非常简单，它是非常具体的指令，没有逻辑运算符，也不支持正则表达式，只能是清晰具体的指令，如删除 HDFS 中文件名为/test.log 文件，这种指令系统中应输入“delete -file /test.log”。这是最基础的指令，Rule 指令是基于此来实现的。该模块的处理流程图如 3-4 所示。

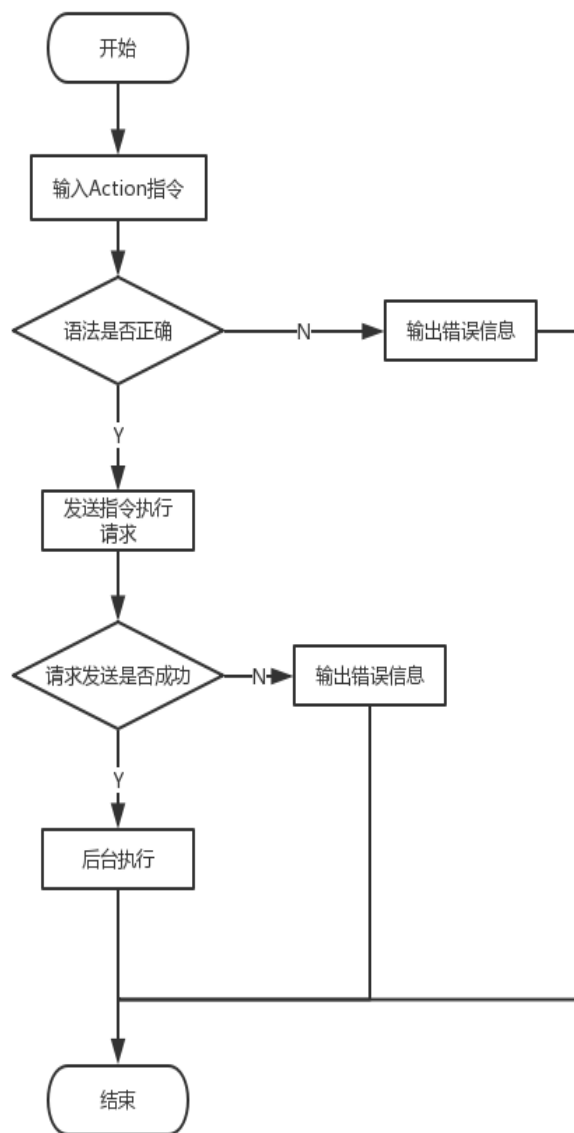


图 3-4 Action 指令流程图

2) Rule 指令设计

Rule 指令较为复杂，它几乎涵盖了系统所有的功能，它定义了任务中哪些文件元数据被涉及，在何种情况下，在何时触发。通过使用 Rule 指令，系统管理人员可以轻松管理整个集群。Rule 指令大体可以分为四个部分：操作对象，触发时机，条件，命令。每个部分可以用“|”和“:”进行分割。下面是对每个部分的具体介绍。

(1) 操作对象如表 3-1 所示。

华中科技大学硕士学位论文

表 3-1 操作对象表

对象	说明	举例
file	HDFS 中的文件	file with path matches /f*.dat"

(2) 触发时机是 Rule 运行时间点，如表 3-2 所示。

表 3-2 触发时机表

格式	说明	举例
at<time>	在指定时间运行 Rule	at "2017-07-2923:00:00" at now
every<time interval>	每隔一段时间运行 Rule	every 1min
from<time> to <time>	与 every 一起用，限定时间段运行 Rule	every 1day from now every 1min from now to now + 7day

(3) 条件是操作对象需要符合的条件，如表 3-3 所示。

表 3-3 条件表

因素	说明	举例
操作对象属性	详细说明在表 3-4 中	length > 5MB
时间	yyyy-MM-dd HH:mm:ss:ms 预定义时间 时间+时间段	"2017-07-29 23:00:00" now now + 7day
时间段	数字+时间单位 某一时间——某一时间 时间段+时间段	5ms, 5sec, 5min, 5hour, 5day now-"2016-03-19 23:00:00" 5hour + 5min
文件大小	数字加文件大小单位	5B, 5kb, 5MB, 5GB, 5TB, 5PB
字符串	支持转义字符	"abc", "123", "Hello world\n"
逻辑运算符	and, or, not	
数字运算符	+, -, *, /, %	
比较	>, >=, <, <=, ==, !=	

(4) 操作对象属性如表 3-4 所示。

表 3-4 对象属性表

操作对象	属性	描述
文件	age	文件上次更改到现在的时间
	atime	上次访问时间
	blocksize	文件块的大小
	inCache	文件在内存中
	isDir	是否是文件夹
	length	文件大小
	path	文件路径
	mtime	上次被修改时间
	unsynced	文件没有同步
	storagePolicy	文件存储策略
	accessCount(Time Interval)	上个时间段文件被访问次数
	accessCountTop(interval,N)	上个时间段访问次数最高的 N 个文件
	accessCountBottom(interval,N)	上个时间段访问次数最低的 N 文件
	accessCountTopOnStoragePolicy(interval,N,\$StoragePolicy")	基于某种存储策略的文件， 上个时间段被访问次数最多 的 N 个文件
	accessCountBottomOnStoragePolicy(interval,N,\$StoragePolicy")	基于某种存储策略的文件， 上个时间段被访问次数最少 的 N 个文件

(5) 命令是具体的操作，如表 3-5 所示。

表 3-5 命令表

命令	说明
allssd	将文件的所有 block 移动到 ssd 上
alldisk	将文件的所有 block 移动到 disk 上
append	向文件添加内容
archive	将文件归档
cache	将文件加载到内存
checkstorage	检查文件存储策略
compact	整合小文件
concat	将某文件夹下文件整合
copy	复制文件
copy2s3	向 AWS S3 中备份文件
delete	删除文件
echo	输出信息
list	列出文件夹下的内容
onedisk	将文件的一个 block 备份移动到 disk 上
onessd	将文件的一个 block 备份移动到 ssd 上
ramdisk	将文件的加载到内存中
read	读文件
rename	修改文件名
truncate	截指定大小的文件
truncate0	将文件大小置为 0B
uncache	将在内存中的文件取消
uncompact	将整合的小文件在重新写入 HDFS
write	写文件
sleep	中断某一段时间
sync	备份文件到另一个集群
user defined actions	提供接口给用户自定义命令

“file with path matches `"/fooA/*.dat": age > 30day | archive`”这就是一条 Rule 指令，它的含义就是在 `/fooA/` 文件夹下，以 `.dat` 为结尾，并且离上一次修改时间超过 30 天的文件，将其归档。利用这种 Rule 指令来管理集群是非常方便的，Rule 指令的处理流程如图 3-5 所示。

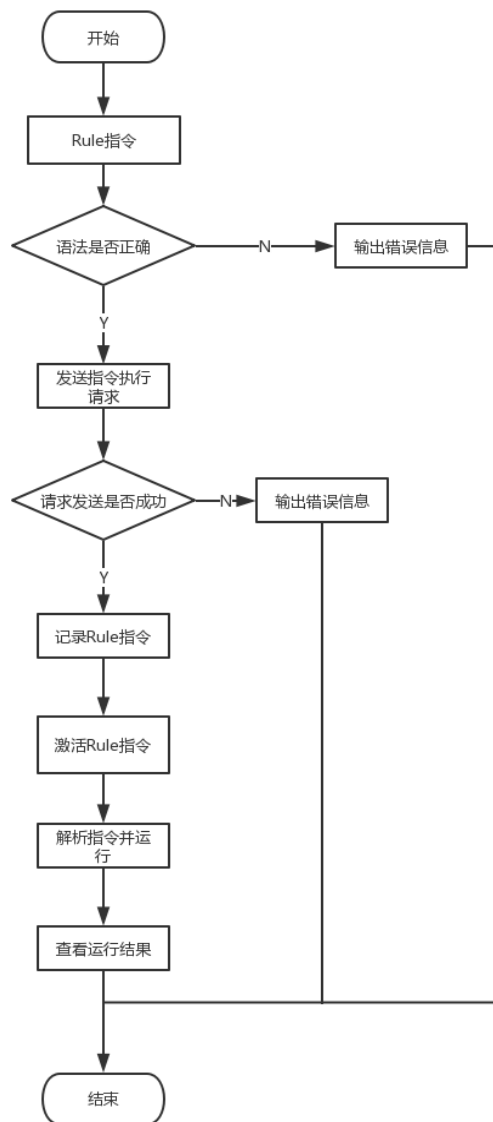


图 3-5 Rule 指令流程图

不同的指令有不同的功能，下面是几个重要的功能设计：

1) 冷热文件存储模块设计

因 HDFS 中文件冷热程度不同，所以要对冷热不同的文件进行不同介质的存储。本系统支持 allssd, alldisk, onessd, onedisk, ramdisk, archive 的存储方案，这几种方案已经在表 3-5 中描述过。文件冷热的程度也是用户根据自己的需求进行判断的，系统提供了基于 accessCount 命令的判断方法，通过判定文件在某一个时间段内被访问过的次数，来判定文件冷热。

本模块最重要的就是获取 NameNode 上集群的所有文件的访问事件，然后将事件信息存储到 MySQL 数据库中，程序根据这些访问记录来判定文件的冷热程度，然后执行文件块的移动。这里有一个关键的设计，就是对于大量的集群访问记录，不应就直接将这条记录直接存储在数据库中，因为随着使用时间的增加，数据库可能存储不了这么庞大的数据量，数据库的检索速度也会变慢，影响系统的可靠性。应该将访问记录按时间段进行统计整合，越靠近现在的时间，时间段间隔应该越小。

对于文件块的移动，应该遵循就近移动的原则，比如要将一个存储在 Disk 上的文件块，移动到 SSD 上。系统会先检查本机有没有 SSD 存储介质，如果有，直接在本机移动，如果本机没有 SSD 存储介质，优先选择同一个机架上的机器，如果同一个机架没有机器有 SSD 存储介质，那就在当前的数据中心寻找，如果再没有，就需要跨数据中心找符合条件的机器。对与冷热文件存储指令，“file : accessCount(1min) > 0 and path matches "/demoCold2Hot/*" | allssd”它的具体含义就是：在最近一分钟内被访问过，而且文件名匹配/demoCold2Hot/*的文件，会被移动到 SSD 上。

本模块的处理流程如图 3-6 所示。

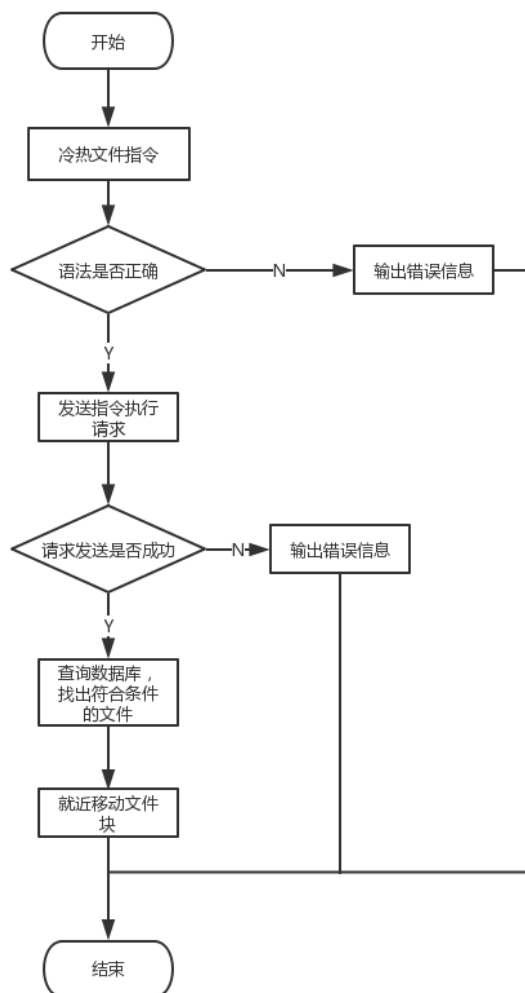


图 3-6 冷热文件存储流程图

2) 小文件整合模块设计

本模块会将用户指定的小文件都写到一个文件容器中，依然支持小文件的读，但是不支持小文件的修改。而且在 HDFS 集群中，小文件会被删除，读取小文件，其实是通过小文件在文件容器的索引找到小文件的信息。索引信息放在数据库中，存放的是文件大小以及文件在文件容器中的位置偏移量。小文件整合的示意图如图 3-7 所示。

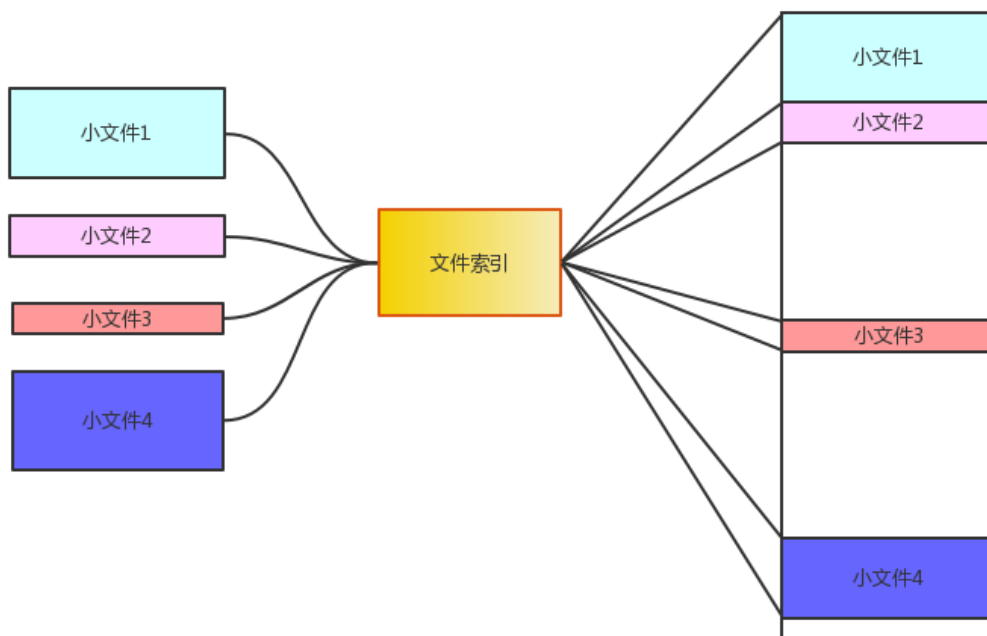


图 3-7 小文件整合示意图

小文件不仅仅可以整合，本系统也支持将整合好的小文件，重新写回 HDFS 集群中去。本模块的处理流程如图 3-8 所示。

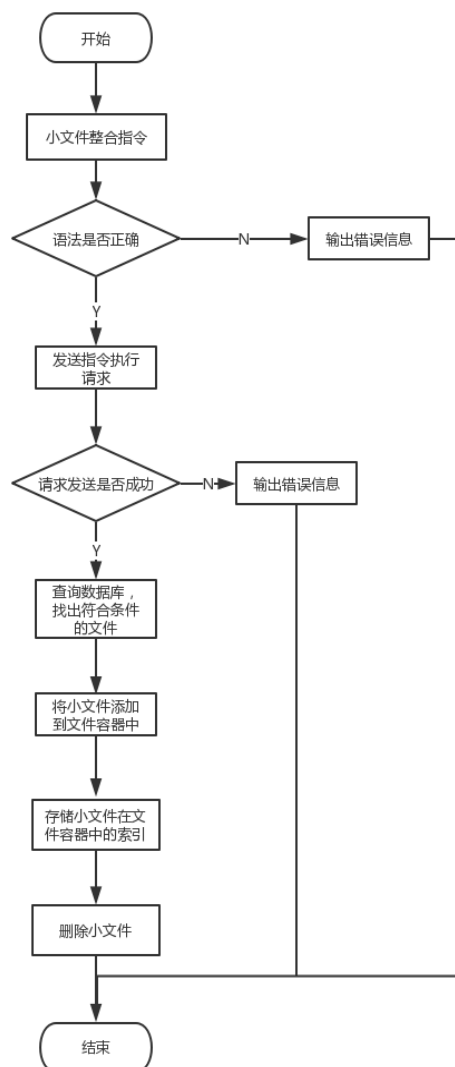


图 3-8 小文件整合流程图

3) 集群数据备份模块设计

集群数据备份模块，为防止运维人员的误操作或黑客的恶意攻击导致灾难性后果，将本集群的文件备份到另一个集群或者云计算存储服务中。数据备份分为同步备份和异步备份两种。同步备份在向集群中写文件时也要同时向另一个集群或者云计算存储写文件，例如指令“`file.path matches /test/*.log | sync_backup clusterB`”。异步备份是通过指令，将集群中已经存在的文件，异步备份。例如指令“`file.path matches`

/test/*.log | async_backup clusterB”。对于读取文件可以指定去集群去读，比如“file.path matches /test/*.log | read clusterB”。也可以随机选择集群进行读取任务，比如“file.path matches /test/*.log | read balance”。

模块的具体流程如图 3-9 所示。

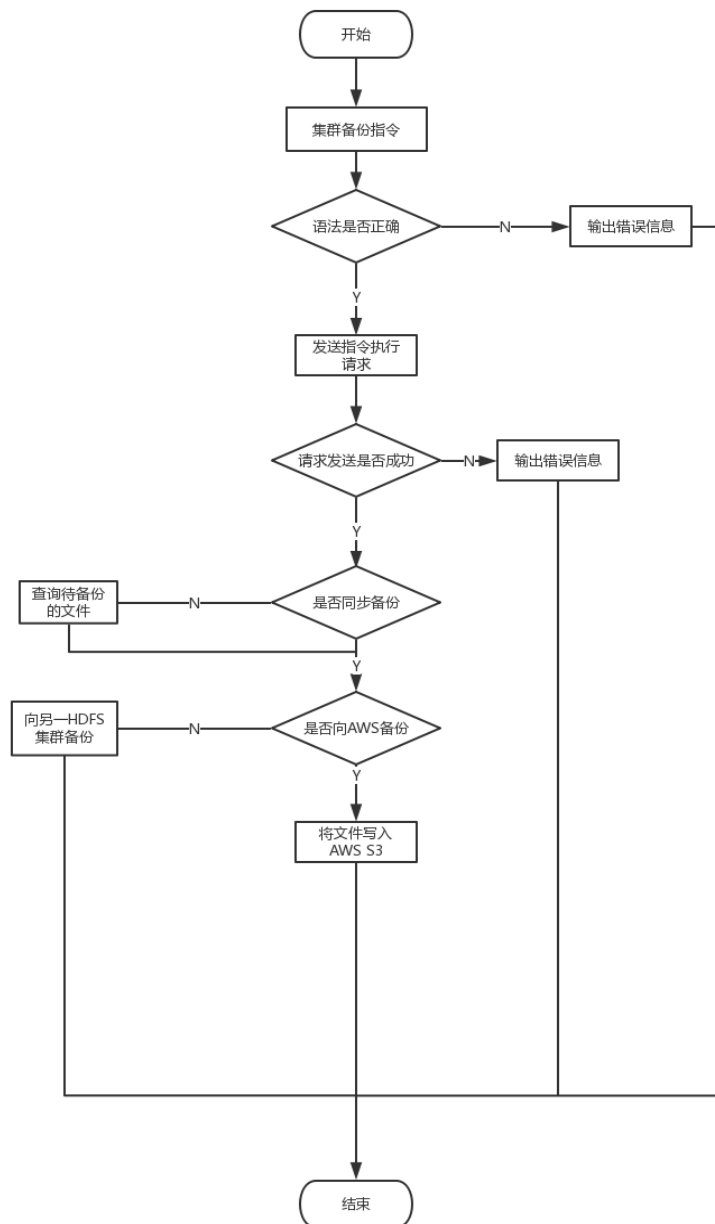


图 3-9 集群备份流程图

3.2.2 任务结果展示及状态管理设计

1) Action 指令的结果展示

Action 指令作为最基本的指令，其结果展示也非常简单，系统提供一个专门展示 Action 指令结果的页面，每一个 Action 指令的结果都占据一行，主要展示内容有 Action 的序号，任务是否成功（成功的指令为绿色，不成功则为红色），任务开始时间，结束时间，任务耗时（单位是毫秒），任务的输出信息（如果任务失败，输出错误日志）。因为执行的 Action 指令可能非常多，所以默认展示最新的任务结果，提供指令结果查询功能，可以按照 Action 序号，Action 任务开始时间，结束时间，或者 Action 类型进行查询。Action 指令结果展示流程处理如图 3-10 所示。

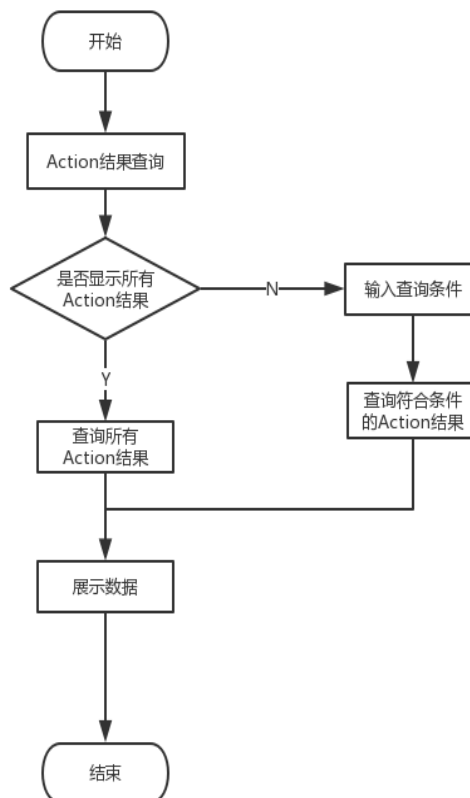


图 3-10 Action 指令结果展示流程图

2) Rule 指令管理及结果查询

Rule 指令是有状态的，它并不像 Action 指令，执行完就结束了。它可能不断执行，或者说在某个时间段不断执行的，Rule 有四种状态，分别是 active, disabled, finished, deleted。这几种状态是会相互转化的，其转化关系如图 3-11 所示。

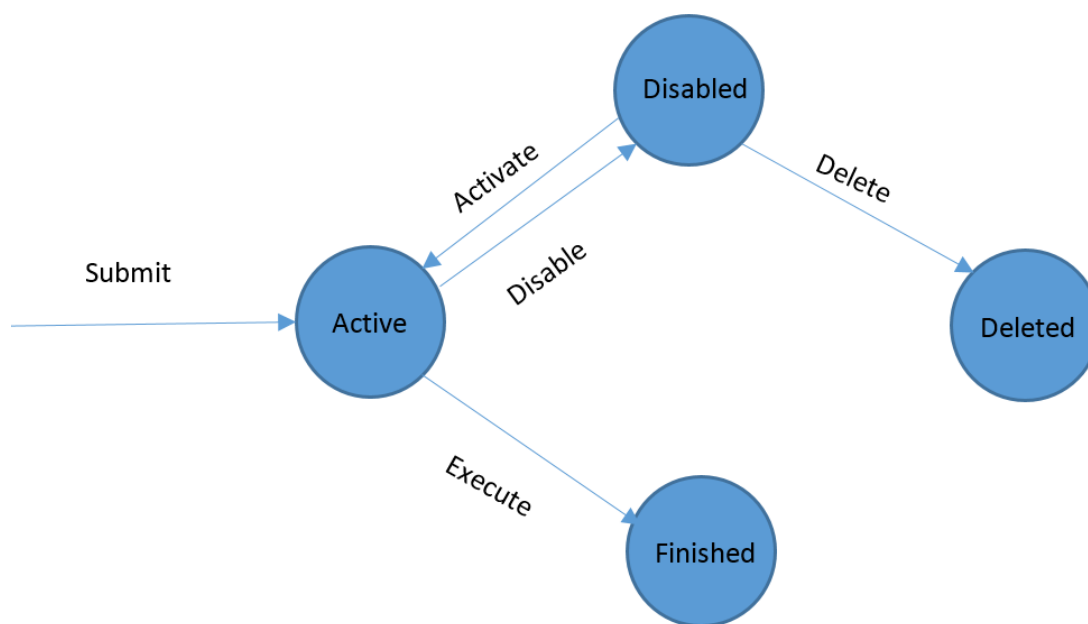


图 3-11 Rule 状态转化图

系统提供 Rule 的提交，激活，暂停，删除的状态管理功能。提交一条 Rule 指令，如果语法正确，会提交成功，在前端界面会显示已经提交的 Rule 指令文本。如果想重新提交，只要点击相应的 Rule 指令就会重新提交。所有提交过的 Rule 指令，都会出现在一个 Rule 指令管理的一个界面，每条 Rule 指令占据一行，显示的内容主要有 Rule 的序号，Rule 的起止时间，Rule 状态变更按钮，删除按钮，以及展示 Rule 产生的结果的详细信息。

3.3 系统数据库设计

系统采用的是 MySQL 5.7 数据库^[30]，因为从使用成本考虑，使用了免费的 MySQL 数据库，而且本系统的数据量也不大，不需要使用 Oracle 数据库。

本系统涉及到的数据主要分为两部分，一部分集群本身的数据，包括 NameNode

的元数据，以及集群产生的事件数据等。另一部分是本系统产生的数据，主要包括指令数据以及指令衍生出来的数据，系统运行所依赖的一些数据也会存在数据库中。这些数据相互关联，比如一条 Rule 记录可能对应上千条的 Action 记录。所以数据库设计，既要满足功能需求，又应该避免冗余，扩展性也应在考虑之中。所以整体的数据库设计非常重要。

3.3.1 数据库 E-R 图

对系统存在的实体进行分析，根据功能需求设计相应的数据库表，针对实体间的关联关系，画出了本系统主要数据库表的 E-R 图，如图 3-12 所示。

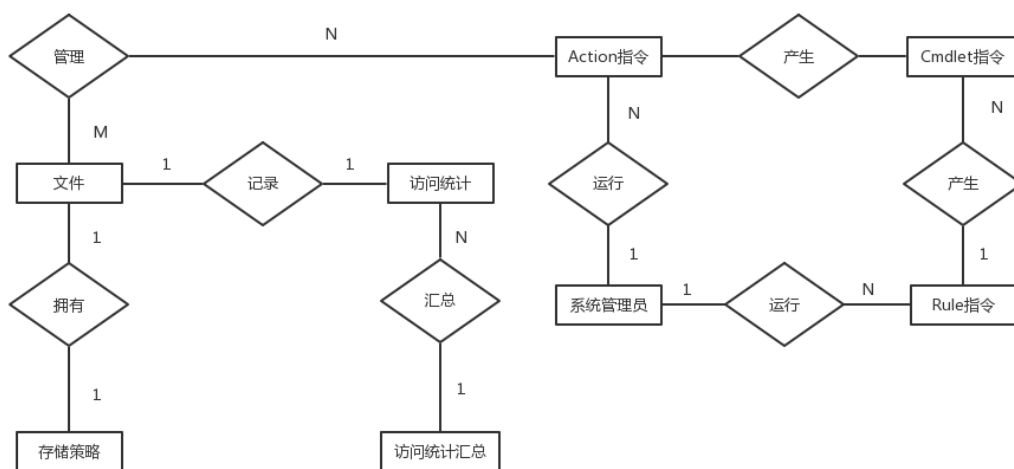


图 3-12 系统数据库 E-R 图

3.3.2 数据库表设计

上一节是对整个数据库的整体设计以及数据库表之间的关系说明，本节是对几个重要数据库表的详细信息进行介绍。

1) 文件信息表

文件信息是从 NameNode 获得的集群的元数据信息，以及本系统对文件的一些附加信息，比如冷热存储策略等字段。文件信息表的详细设计如表 3-6 所示。

表 3-6 文件信息表

字段名	数据类型	能否为空	描述	备注
path	varchar(4096)	否	文件目录	
fid	bigint(20)	否	文件主键	主键
length	bigint(20)	否	文件大小	
block_replication	smallint(6)	否	文件块备份个数	
block_size	bigint(20)	否	文件块大小	
modification_time	bigint(20)	否	文件最后修改时间	
access_time	bigint(20)	是	文件访问时间	
is_dir	bit(1)	否	是否是文件夹	
sid	tinyint(4)	否	存储策略 ID	外键
oid	smallint(6)	否	所属者 ID	外键
gid	smallint(6)	否	所属组 ID	外键
ec_policy_id	smallint(6)	是	纠删码策略 ID	外键
permission	smallint(6)	否	权限	

2) 存储策略表

存储策略即对冷热文件的不同存储方式，存储策略表的详细设计见表 3-7 所示。

表 3-7 存储策略表

字段名	数据类型	能否为空	描述	备注
sid	tinyint(4)	否	存储策略 ID	主键
policy_name	varchar(64)	否	存储策略名称	

3) 加载到内存的文件表

表 3-8 内存文件表

字段名	数据类型	能否为空	描述	备注
fid	bigint(20)	否	文件 ID	主键
from_time	bigint(20)	否	加载到内存时间	
last_access_time	bigint(20)	否	上一次访问时间	
num_accessed	int(11)	否	被访问的次数	

4) 秒级别, 分钟级别, 小时级别, 天级别, 月级别, 年级别访问统计表

系统会记录用户的所有访问信息, 并且会对某个时间段的信息进行汇总, 根据时间段的长度不同, 表的级别有秒级别, 分钟级别, 小时级别, 天级别, 月级别, 年级别, 每个时间段都存放文件 ID 以及访问的次数。详细表设计见表 3-9。

表 3-9 不同时间段访问统计表

字段名	数据类型	能否为空	描述	备注
fid	bigint(20)	否	文件 ID	主键
count	int(11)	否	访问次数	

5) 时间段统计表汇总表

系统会存储不同时间段的访问统计表, 此表是这些统计表的汇总。比如秒级别的表, 系统可能会存在多个, 那么这些表的表名, 起止时间都会存在这个汇总表中。汇总表的详细设计如表 3-10 所示。

表 3-10 汇总表

字段名	数据类型	能否为空	描述	备注
table_name	varchar(255)	否	表名	
start_time	bigint(20)	否	表的开始时间	
end_time	bigint(20)	否	表的结束时间	

6) Rule 表

系统最重要的指令是 Rule 指令,每次提交 Rule 指令都会将指令存到数据库当中, Rule 指令的状态,提交时间等信息也会存在数据库当中,详细的表结构设计如表 3-11 所示。

表 3-11 Rule 表

字段名	数据类型	能否为空	描述	备注
id	int	否	Rule ID	主键
name	varchar(255)	是	Rule 名称	
state	tinyint(4)	否	状态	
rule_text	varchar(4096)	否	命令内容	
submit_time	bigint(20)	否	提交时间	
last_check_time	bigint(20)	是	上次检查时间	
checked_count	int(11)	否	检查次数	
generated_cmdlets	int(11)	否	产生的命令数目	

7) Cmdlet 表

Cmdlet 是由 Rule 产生的命令,一个 Rule 可以产生多个 Cmdlet,一个 Cmdlet 可以产生多个 Action 指令。所以 Rule 产生的 Cmdlet 会存储在数据库中,此表的详细设计如表 3-12 所示。

表 3-12 Cmdlet 表

字段名	数据类型	能否为空	描述	备注
cid	int(11)	否	Cmdlet ID	主键
rid	int(11)	否	Rule ID	
aids	varchar(4096)	否	多个 Action ID	

续表 3-12 Cmdlet 表

state	tinyint(4)	否	状态	
parameters	varchar(4096)	否	参数	
generate_time	bigint(20)	否	产生时间	
state_change_time	bigint(20)	否	状态改变时间	

8) Action 表

Action 指令是最基础的指令，提交 Action 指令都会存在数据库中，具体的表结构设计如表 3-13 所示。

表 3-13 Action 表

字段名	数据类型	能否为空	描述	备注
aid	int(11)	否	Action ID	主键
cid	int(11)	否	Cmdlet ID	
action_name	varchar(4096)	否	名称	
args	varchar(4096)	否	参数	
result	text	否	结果	
log	text	否	日志	
successful	tinyint(4)	否	是否成功	
create_time	bigint(20)	否	创建时间	
finished	tinyint(4)	否	是否结束	
finish_time	bigint(20)	否	结束时间	
progress	int(11)	否	进度	

3.4 本章小结

本章主要对系统进行设计，设计目标即是满足需求，针对需求分析，系统设计首先从整体架构进行，之后对每个模块的功能进行了详细设计，并画出了相应的流程图。数据库设计也按照数据库设计范式进行设计，避免冗余，提高扩展性。不仅给出了数据库 E-R 图，对重要数据库表给出了详细设计。通过本章从整体到细节的设计，为系统实现指明了方向，也提供了清晰的思路。

4 系统实现

本系统整体采用 B/S 架构。前端界面负责输入指令以及结果展示，采用 Zeppelin 作为前端框架，其提供的基于网页的 notebook 方便用户输入指令，Zeppelin 也提供了数据分析、数据可视化等功能，而且提供 RESTful 接口方便与后端进行信息传输。后端负责响应请求，解析指令，返回结果等功能，采用 Jersey 作为 RESTful 服务框架，系统为方便操纵数据库，也采用 Spring 框架，但仅仅是使用 Spring 中的数据库模板，并未使用其 IOC，AOP 等功能。为提高数据库连接速度，使用 Druid 数据库连接池^[31]。Jersey 也方便与 Spring 进行整合。为实现系统的分布式运行使用 Akka 框架进行开发。在具体实现部分，本章对重要功能模块进行详细的实现介绍，包括系统界面展示，关键代码解读，以及重要技术点进行阐述。

4.1 开发环境与工具

本系统是在 Ubuntu 16.04 系统中进行开发，使用 Java，Scala，JavaScript 编程语言进行开发，借助 Hadoop，Hazelcast 等组件，使用 MySQL 数据库完成功能实现。因本系统是多人开发，也是用 Git 2.14.1 进行代码托管，实现多人可在不同分支进行开发，所有代码都归并到主分支上，从而提高开发效率。具体的开发语言，开发工具以及开源组件的使用如表 4-1 所示。

表 4-1 开发环境与工具表

分类	条目	开发环境或工具信息
编程语言	编程语言	Java, Scala, JavaScript, Python
开发环境	数据库	MySQL 5.7
	JDK	JDK 1.8
	Web 容器	Jetty
	操作系统	Ubuntu 16.04

续表 4-1 开发环境与工具表

	IDE	intellij idea
开源组件	Hadoop	Hadoop 2.7.3
	Hazelcast	Hazelcast 3.6
	Jersey	Jersey 2.6
硬件配置	CPU	i3-7100T
	内存	12GB
	SSD	256GB
	带宽	10M
辅助工具	Git	Git 2.14.1

表 4-1 中的开发环境是保证稳定的前提下较新的版本，因本系统主要应用于 Linux 系统，所以开发工作也在 Linux 环境中进行。大部分的后端开发使用 Java 语言，采用了业界最受欢迎的 Java IDE intellij idea，它不仅方便 Java 开发，调试，而且集成了 Git，使用此 IDE 非常方便进行开发工作。在开发过程中，团队使用 Git 来托管代码，每个成员都在自己的分支上进行开发工作，不断的合并到 trunk 分支上。本系统代码编写十分规范，运用多种设计模式，注释清晰易懂，而且对于大部分功能都会编写测试用例，每次提交代码都会进行代码审查以及测试用例是否通过。

4.2 系统部署与配置

本系统涉及的设备角色比较多，主要有 Hadoop 集群，备份集群，MySQL 服务器，Active Server，Standby Server，多个 Agent Server，以及界面访问计算机。下面对这几种设备进行介绍。

Hadoop 集群：本系统是在 Hadoop 集群之上的管理系统，所以此集群主要用来存储分析数据以及本系统的请求，而且此系统一般规模较大，设备数可能会达到上百个。

备份集群：因系统需要备份，所以还应该有备份集群存在，当然如果是使用 AWS

S3 则不需要另一个备份集群。

MySQL 服务器：MySQL 进程运行的服务器。

Active Server：运行状态的系统程序服务器。

Standby Server：是用来抗单节点故障的服务器^[32]。

Agent Server：Active Server 会将任务分发给 Agent 服务器来提高并发度。

设备的配置说明如下表 4-2。

表 4-2 系统配置表

设备分类	硬件配置要求	软件配置要求	运行程序
Hadoop 集群	内存 64GB 及以上 CPU 16 核及以上 硬盘 2TB 及以上 服务器个数 10 个及以上	CentOS 6.0 及以上 Debian 6.0 及以上	Hadoop2.7.3 Zookeeper3.4.8 JDK1.8
备份集群	内存 16GB 及以上 CPU 4 核及以上 硬盘 2TB 及以上 服务器个数 5 个及以上	CentOS 6.0 及以上 Debian 6.0 及以上	Hadoop2.7.3 Zookeeper3.4.8 JDK1.8
MySQL 服务器	内存 64GB 及以上 固态硬盘 256GB 及以上 CPU 32 核及以上	Linux 操作系统 MySQL 5.6 及以上	MySQL 5.6 及以上
Active Server	内存 128GB 及以上 CPU 32 核及以上 硬盘 2TB 及以上	CentOS 6.0 及以上 Debian 6.0 及以上	系统后端服务程序 Hazelcast 2.0
Standby Server	内存 32GB 及以上 CPU 16 核及以上 硬盘 2TB 及以上	CentOS 6.0 及以上 Debian 6.0 及以上	系统后端服务程序 Hazelcast 2.0
Agent Server	内存 128GB 及以上 CPU 32 核及以上 硬盘 2TB 及以上	CentOS 6.0 及以上 Debian 6.0 及以上	系统后端服务程序
前端界面访问机器	内存 4GB 及以上 CPU 1 核及以上 硬盘 128GB 及以上	Windows XP 及以上 Mac OS10.10 及以上 Linux 操作系统	浏览器（Chrome，火狐，IE）

4.3 系统主要功能实现

本系统的主要功能主要有两大块，一是智能运维指令的提交与执行，二是运行结果展示。本小节主要针对这两大块内容进行实现阐述，包括页面展示，逻辑流程以及关键代码展示。

4.3.1 智能运维指令模块实现

本模块是系统最核心的模块，运维指令分为 Action 指令以及 Rule 指令两种。

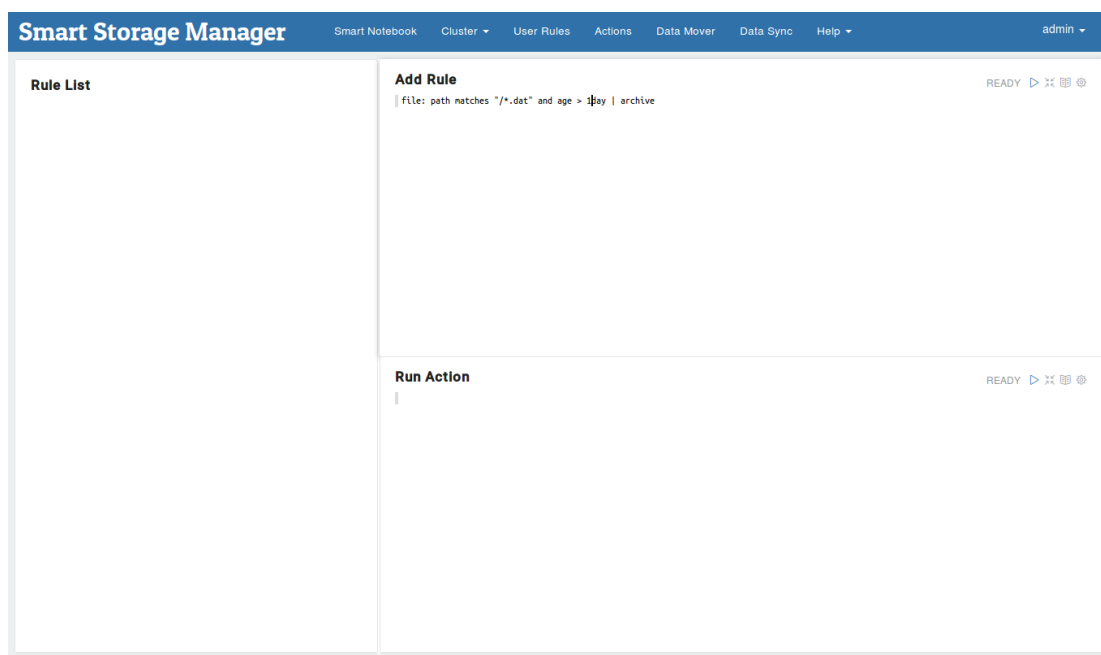


图 4-1 指令输入界面

1) Action 指令实现

用户通过提交 Action 指令来实现各种运维操作。用户在前端界面输入指令，并点击运行指令按钮就会执行本命令。下面是详细的实现过程。

表现层：系统使用 Zeppelin 框架，提供了 notebook 来输入指令，点击运行按钮会将需要运行的指令以 json 字符串的形式发给后端进行解析，如果指令输入有误，会有提示。

业务逻辑层：Jersey 的拦截器会对不同类型的请求进行分发，对应的业务处理类会首先对指令进行解析，如果指令输入错误则直接返回错误信息。如果指令解析成功，会将其封装成一个任务分发给 Agent^[33]，任务开始时，首先会调用持久层的方法将此 Action 命令存入数据库中，然后查询指令需要的数据库中的数据，利用这些数据系统可以完成对 Hadoop 集群的具体操作，成功后将数据库中指令的状态设置为成功即可。

持久层：本层主要封装对数据库操作的方法，方便业务层的调用。对数据库的增删改查主要利用 Spring 的 JdbcTemplate 进行操作。

以上就是一个 Action 指令执行的全部过程，Action 指令可以完成很多功能，比如小文件整合，集群备份等，下面对几个主要功能进行详细的实现阐述。

（1）冷热文件的不同存储策略模块实现

本模块主要实现目标就是将冷热不同的文件块进行移动，首先要解决的问题是如何判断文件的冷热程度。判断文件的冷热，需要根据文件被用户访问的频率来进行判定，所以系统需要记录用户的文件访问记录。Hadoop 提供了 Inotify 机制来获取用户访问集群的事件信息。

通过 `getInotifyEventStream(long lastReadTxid)` 方法可以获取大于事件 id 值 `lastReadTxid` 的所有事件，系统专门有线程进行定时调用 `getInotifyEventStream` 方法，得到所需要的事件信息，并且保存所得到的最近事件的 id 值到数据库中，方便下一次线程调用 `getInotifyEventStream` 方法。得到的事件信息，通过解析，判断事件类型，并对 MySQL 数据库中集群元数据进行更改，以达到集群元数据与数据库中元数据同步的效果。每个事件都是一次访问，所以应该被记录下来，如果仅仅对每一个新得到的事件，都向数据库中插入一条访问记录，那么随着事件的不断增多，会带来两个问题。一是海量的记录信息占据磁盘空间，另外，对冷热文件进行判断时，因数据量非常庞大，数据的统计可能会很慢，也会影响系统的稳定性。

为解决事件信息存储及统计的问题，本系统采用多级别表汇总统计的方法。系统中存在秒级别的表（表的起止时间相差是 5 秒钟），分钟级别的表（表的起止时间相差是 5 分钟），小时级别的表（表的起止时间相差是一小时），天级别的表（表

的起止时间相差是一天），月级别的表（表的起止时间相差是 30 天），年级别的表（表的起止时间相差是一年）。不同级别的表存放的数据是一样的，都是文件 id 以及在此表的起止时间段内被访问的次数。而且各个级别的表的个数不相同（可以自己设置），但是表的起止时间相连。比如系统中存放不能超过 20 张秒级别的表，不能超过 10 张分钟级别的表。

同级别的多个表构成一个队列 `AccessCountTableDeque`，随着时间的增加，表的个数会增加，比如每隔 5 秒钟就要新建一张秒级别的表。当表的个数超过限制，就要将表队列中队头的表中数据向更高级别的表去汇总，并删掉队头的那张表。比如，当有一条新的记录插入数据库，肯定先往秒级别的表插入，首先会判断当前事件的时间是否在秒级表队列队尾那张表的起止时间之中，如果在起止时间之内，直接向表中插入数据即可。如果不在表的起止时间之内，就要在队尾新添加一张表，但是新添了一张表，可能超过了表队列中的最大个数，所以每个队列都会设置一个监听器，在增加表的时候，如果表的数量超出限制，监听器会把队头的那张表的数据全部汇总到分钟级别的表中，汇总完会删掉队头的表，如此一来保证队列中的表的个数不会超过限制个数。同理，如果在向分钟级别的表中插入数据的时候，可能也需要新建一张分钟级别的表，也要给分钟级别的表队列设置监听器，来保证队列中的表不会超过最大限制。以此类推，天级别的表队列，月级别的表队列都是这个原理。

通过这种实现方式，每个表中的数据其实就是当前表的起止时间段的汇总结果，这样就解决了事件记录过多，汇总难度过大的问题，但是这种解决方案也损失了一些精度，因为越靠近系统当前使用时间，时间段越小，统计记录就越精确，比如查询最近 20 秒钟的文件访问情况，那么我们可以在通过秒级别表的队列中查到精确结果，但是如果要查询最近一个月的文件访问情况，统计结果与真实数据是有出入的，但是文件冷热程度的判断精度要求并不需要分毫不差，从系统的稳定性，可用性考虑，精度显得不是那么重要。添加一张表的关键代码如下：

```
public boolean addAndNotifyListener(AccessCountTable table) {  
    if (!this.isEmpty()) {  
        assert table.getEndTime() > this.peekLast().getEndTime();  
    }  
}
```

```
    }  
    super.add(table);//add a table  
    if (this.listener != null) {  
        this.listener.tableAdded(this, table);//notify listener  
    }  
    tableEvictor.evictTables(this, this.size());  
    return true;  
}
```

当系统后端接收到文件存储策略的指令，比如将最近一天内访问次数超过 5 次的文件移动到 SSD 上，首先会解析指令，查询数据库访问记录，得到所有符合条件的所有文件，对每个文件的块进行移动。集群的机器有远近之分，根据网络拓扑就够，文件块会就近移动，系统的 movePlan 类来完成此功能。关键代码如下：

```
boolean scheduleMoveReplica(DBlock db, Source source,  
    List<String> targetTypes) {  
    // Match storage on the same node  
    if (chooseTargetInSameNode(db, source, targetTypes)) {  
        return true;  
    }  
    if (networkTopology.isNodeGroupAware()) {  
        if(chooseTarget(db,source, targetTypes, Matcher.SAME_NODE_GROUP))  
            {return true; }  
    }  
    // Then, match nodes on the same rack  
    if (chooseTarget(db, source, targetTypes, Matcher.SAME_RACK)) {  
        return true;  
    }  
    // At last, match all remaining nodes
```

```
return chooseTarget(db, source, targetTypes, Matcher.ANY_OTHER);  
}
```

有了文件块的原地址和目标地址，通过 socket 将数据传输到目标地址即可。

(2) 小文件整合模块实现

根据上一章的小文件功能设计，系统对其进行实现。首先后端会解析前端的指令，然后找到需要整合的小文件，依次将小文件写到文件容器中，而且要记录文件在文件容器中的位置偏移量。当文件整合好之后，删除小文件即可。主要代码如下：

```
for (String smallFile : smallFileList) {  
    if ((smallFile != null) && !smallFile.isEmpty() && dfsClient.exists(smallFile))  
    {  
        long fileLen = dfsClient.getFileInfo(smallFile).getLen();  
        if (fileLen > 0) {  
            try (InputStream in = dfsClient.open(smallFile)) {  
                // Copy bytes of small file to container file  
                IOUtils.copyBytes(in, out, 4096);  
                // Truncate small file, add file container info to XAttr  
                CompactFileState compactFileState = new CompactFileState(  
                    smallFile, new FileContainerInfo(containerFile, offset, fileLen));  
                truncateAndSetXAttr(smallFile, compactFileState);  
  
                // Update compact file state map, offset, status, and log  
                compactFileStates.add(compactFileState);  
                offset += fileLen;  
                this.status = (smallFileList.indexOf(smallFile) + 1.0f)  
                    / smallFileList.size();  
                appendLog(String.format(  
                    "Compact %s to %s successfully.", smallFile, containerFile));
```

```
    } catch (IOException e) {  
        // Close output stream and put compact file state map into action  
    }  
}  
}
```

(3) 文件备份

系统可以将文件备份到另一个 Hadoop 集群^[34]，也可以备份到 AWS S3 中，只要在指令中指明备份的目的地就好。备份的方式也有两种，一种是同步备份，向集群写文件的同时也要备份，只有文件在集群和备份端都写成功，才算成功。另一种是异步备份，将集群中已存在的文件备份。主要代码如下：

```
in = getSrcInputStream(src);  
out = getDestOutPutStream(dest, offset);  
//skip offset  
in.skip(offset);  
byte[] buf = new byte[bufferSize]; //buffer Initialization  
long bytesRemaining = length;  
while (bytesRemaining > 0L) {  
    int bytesToRead =  
        (int) (bytesRemaining < (long) buf.length ? bytesRemaining :  
            (long) buf.length);  
  
    int bytesRead = in.read(buf, 0, bytesToRead); //read data from src  
    if (bytesRead == -1) {  
        break;  
    }  
  
    out.write(buf, 0, bytesRead); //write data to dest  
  
    bytesRemaining -= (long) bytesRead;  
}
```

2) Rule 指令模块实现

Rule 指令较为复杂，根据上一章的 Rule 功能设计，它主要有四个部分组成。用户输入指令，并点击运行按钮即可运行 Rule 指令。下面是此模块的详细实现。

表现层：系统使用 Zeppelin 框架，提供了 notebook 来输入指令，点击运行按钮会将需要运行的指令以 json 字符串的形式发给后端进行解析，如果指令输入有误会有提示。如果正确，那么前端界面会将此 Rule 添加到已经提交的 Rule 列表当中。

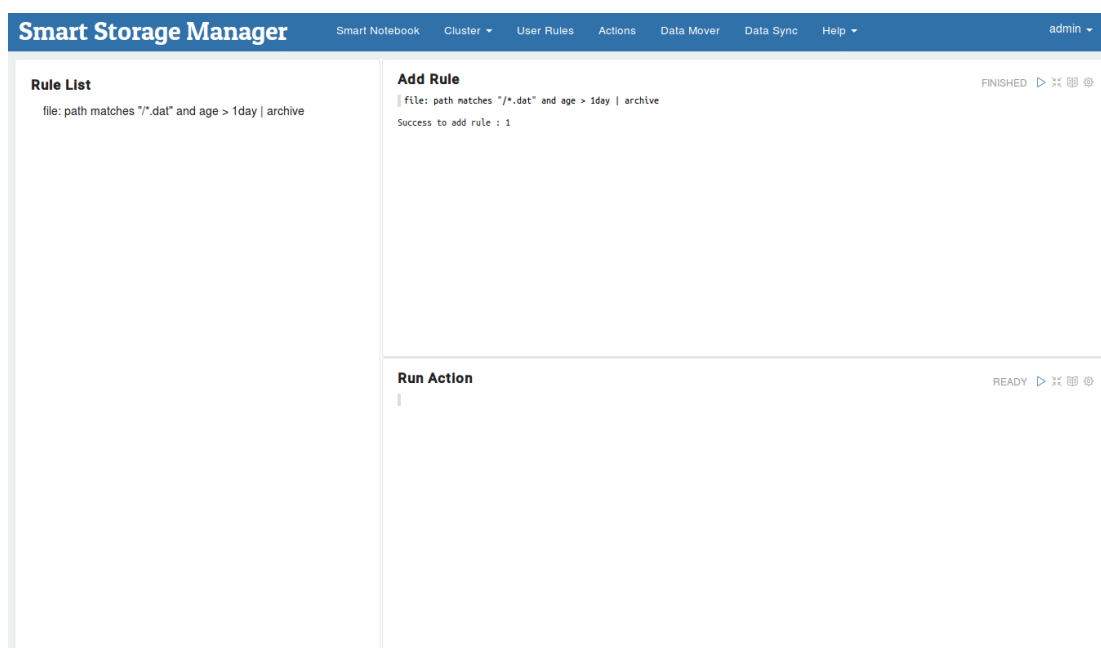


图 4-2 Rule 指令提交界面

业务逻辑层：Jersey 的拦截器会对不同类型的请求进行分发，对应的业务处理类会首先对指令进行解析，如果指令输入错误则直接返回错误日志。如果指令解析成功，首先会将 Rule 指令存入数据库当中。因 Rule 指令可能有定时执行的任务（如每隔一天删除某些文件），所以系统又抽象出来一个概念：Cmdlet。Cmdlet 是在 Rule 指令和 Action 指令之间的桥梁。一个 Rule 指令可能会有多个 Cmdlet，一个 Cmdlet 可能会产生多个 Action 指令。比如指令“file with path matches “/fooA/*.log”| every 1day | age > 30day | archive”，每天都会执行一次，当每天执行的时候都会产生一个 Cmdlet，这个 Cmdlet 的功能就是找到匹配“/fooA/*.log”并且存在时间超过 30 天的文件，并将

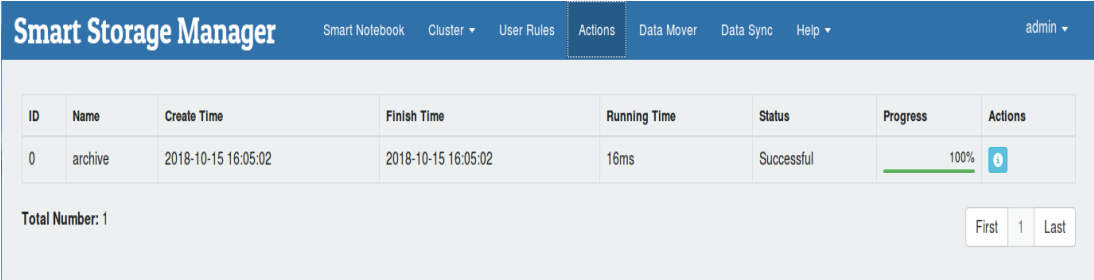
他们归档。此 Cmdlet 又会产生诸多具体执行的 Action 指令。当 Rule 指令运行产生一个 Cmdlet 指令，他会将此 Cmdlet 分发给 Agent 服务器来执行，Cmdlet 指令执行时也会调用持久层的方法将其存在数据库中。然后对 Cmdlet 进行解析，产生一个或多个 Action 指令。如何运行 Action 指令已经在上文介绍过。运行成功后会将结果写入数据库中。

持久层：本层主要封装对数据库操作的方法，方便业务层的调用。对数据库的增删改查主要利用 Spring 的 JdbcTemplate 进行操作。

4.3.2 运行结果展示模块实现

1) Action 指令的结果展示

所有 Action 指令以及 Rule 产生的 Action 指令运行的结果都会显示在 Actions 界面上，每条记录显示的内容包括 ID，Action 任务名称，创建时间，结束时间，运行任务所用的时间，运行状态（有成功和不成功两种状态），以及结果信息。点击结果信息会有具体的任务结果展示。



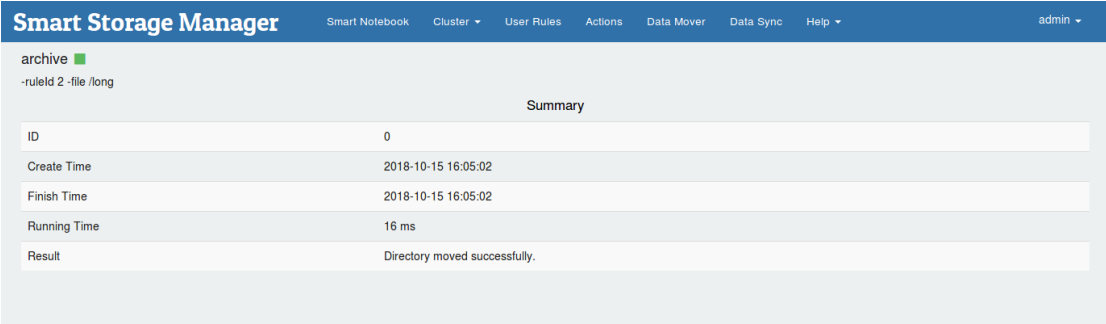
ID	Name	Create Time	Finish Time	Running Time	Status	Progress	Actions
0	archive	2018-10-15 16:05:02	2018-10-15 16:05:02	16ms	Successful	100%	

Total Number: 1

First 1 Last

图 4-3 Action 任务结果展示图

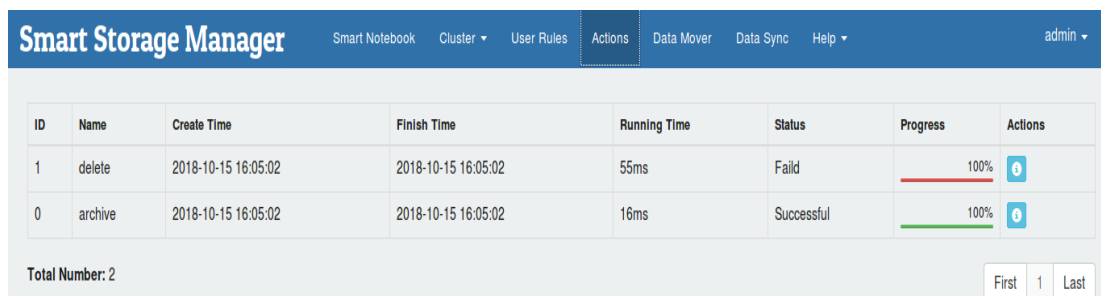
任务的具体结果信息如图 4-4 所示。



Summary	
ID	0
Create Time	2018-10-15 16:05:02
Finish Time	2018-10-15 16:05:02
Running Time	16 ms
Result	Directory moved successfully.

图 4-4 任务具体结果信息展示图

如果有任务失败，显示结果会稍有不同，如图 4-5 所示。



ID	Name	Create Time	Finish Time	Running Time	Status	Progress	Actions
1	delete	2018-10-15 16:05:02	2018-10-15 16:05:02	55ms	Failed	100%	
0	archive	2018-10-15 16:05:02	2018-10-15 16:05:02	16ms	Successful	100%	

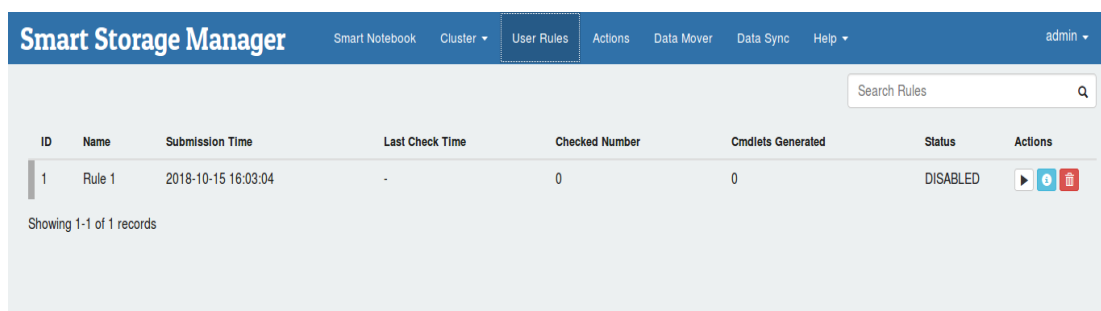
Total Number: 2

First 1 Last

图 4-5 任务结果展示图

2) Rule 状态管理以及结果展示

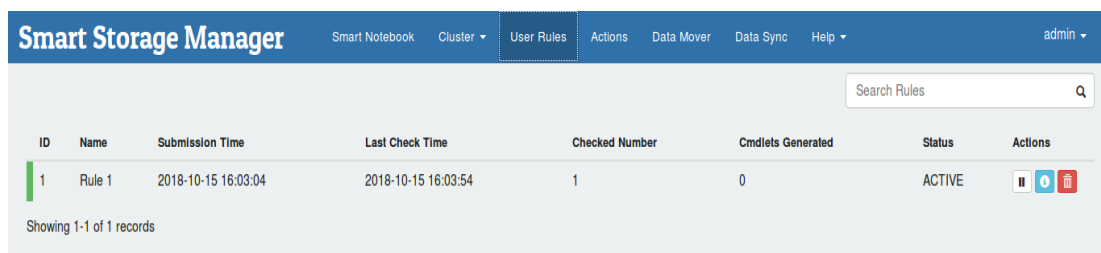
Rule 指令经过提交之后，前端会有专门的界面（User Rules）来管理这些提交成功的指令。提交的 Rule 指令实际上并未真正运行，只有点击运行按钮才会运行，暂停 Rule 任务也是要点击这个按钮。系统也提供 Rule 指令的删除操作。下图是 Rule 状态管理界面。



ID	Name	Submission Time	Last Check Time	Checked Number	Cmdlets Generated	Status	Actions
1	Rule 1	2018-10-15 16:03:04	-	0	0	DISABLED	

Showing 1-1 of 1 records

图 4-6 Rule 未运行状态图



ID	Name	Submission Time	Last Check Time	Checked Number	Cmdlets Generated	Status	Actions
1	Rule 1	2018-10-15 16:03:04	2018-10-15 16:03:54	1	0	ACTIVE	

Showing 1-1 of 1 records

图 4-7 Rule 已运行状态图

系统还提供更加详细的 Rule 指令结果展示功能，点击 Actions 栏目的中间按钮，会显示 Rule 产生的具体结果信息。如下图 4-8 所示。

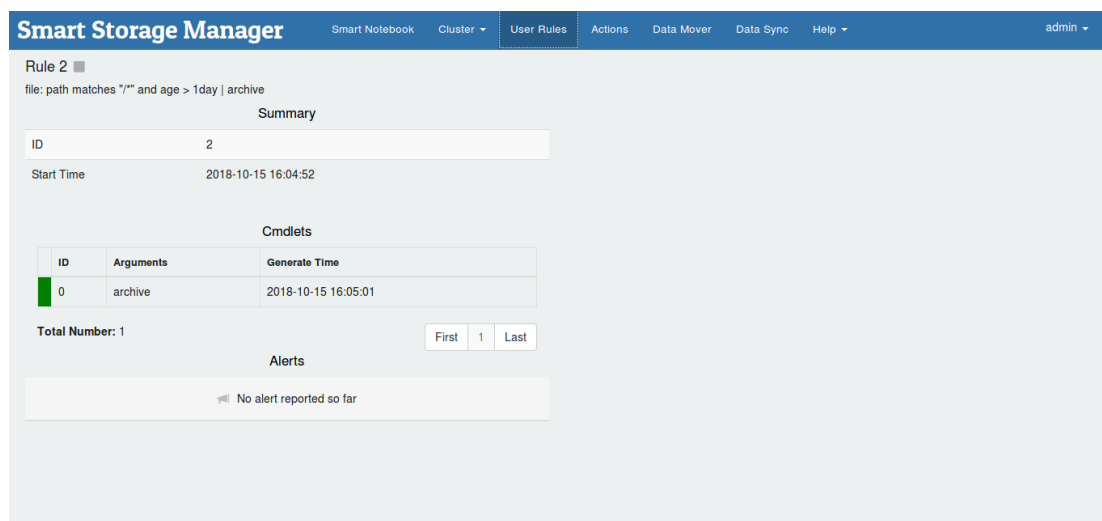


图 4-8 Rule 结果展示图

Rule 结果展示的信息包括 Rule 指令内容，以及产生的 Action 任务。而 Action 任务的结果信息根据 ID 值，可以再 Actions 界面查询获得。

4.4 本章小结

本章是对系统的具体实现的阐述。根据上一章的系统设计，本章从系统的开发环境，以及部署环境进行详细的说明，针对具体功能模块的实现，系统从表现层，业务逻辑层以及持久层进行说明。还给出了不同模块的界面图以及关键代码。模块的实现难度大小不一，对于实现难度大，关键技术点多的模块，比如冷热数据存储模块，本章对其进行了详细的说明，包括原理，技术难点以及优化策略等。

5 系统测试

按照需求分析和系统设计，系统的开发工作已经完成。在开发过程中，已经利用 junit 单元测试框架进行过单元测试。但是系统完成后还没有进行集成测试。为了查看系统是否能达到需求分析中的目标，各个功能模块是否能正常运行，系统的是否稳定可靠，以及发现系统中的问题，会对系统做功能性测试和非功能性测试。

5.1 系统测试环境

系统测试环境应该尽量与生产环境的软硬件配置相同^[35]，系统环境配置如下表所示。

表 5-1 测试软件环境说明表

软件类型	详细配置说明
用户浏览器	IE 8, Chrome 36.0.0.0, 火狐浏览器 63.0
用户操作系统	Win7, Win10, Mac 10.14
Hadoop 集群软件环境	Hadoop 2.7.3, CentOS 6.0 系统, JDK1.8
备份集群软件环境	Hadoop 2.7.3, CentOS 6.0 系统, JDK1.8
Active Server	CentOS 6.0 系统, JDK1.8, Hazelcast 2.0
Standby Server	CentOS 6.0 系统, JDK1.8, Hazelcast 2.0
Agent Server	CentOS 6.0 系统, JDK1.8
数据库服务器	MySQL 5.7

系统硬件测试环境如表 5-2 所示。

表 5-2 测试硬件环境说明表

硬件类型	详细配置说明
客户端硬件配置	内存 4GB，CPU 2 核，硬盘 128GB
Hadoop 集群硬件配置	共 15 台服务器，包括： NameNode 服务器 1 台：内存 64GB，CPU 16 核，硬盘 2TB Secondary NameNode 服务器 1 台：内存 64GB，CPU 16 核，硬盘 2TB DataNode 服务器 13 台：内存 32GB，CPU 8 核，硬盘 4TB
备份集群软件环境	共 10 台服务器，包括： NameNode 服务器 1 台：内存 64GB，CPU 16 核，硬盘 2TB Secondary NameNode 服务器 1 台：内存 64GB，CPU 16 核，硬盘 2TB DataNode 服务器 8 台：内存 32GB，CPU 8 核，硬盘 4TB
Active Server	内存 128GB，CPU 16 核，硬盘 2TB
Standby Server	内存 128GB，CPU 16 核，硬盘 2TB
Agent 服务器	一共 8 台 Agent 服务器：内存 64GB，CPU 16 核，硬盘 2TB
MySQL 服务器	内存 128GB，CPU 16 核，硬盘 2TB

5.2 系统功能测试

为检测系统功能是否运行正常，针对每个功能模块，编写测试用例，测试用例要覆盖系统全部功能，尽可能多的测试系统的功能是否正常，按照测试方案设计测试用例。因测试用例繁杂，下表主要挑选有代表性的测试用例。

表 5-3 测试用例表

主要模块	功能操作	测试目的	测试操作步骤	预期效果	实际效果
Action 指令 模块	小文件整合	验证不指定文件容器，小文件整合是否成功	1.在网页中输入将 10 个文件整合的指令 2.点击运行按钮	Action 界面会显示此指令运行成功	与预期效果一致
		验证指定文件容器，小文件整合是否成功	1.在网页中输入将 10 个文件整合到指定文件的指令 2.点击运行按钮	Action 界面会显示此指令运行成功	与预期效果一致
	冷热数据存储	验证将文件加载到内存中是否成功	1.在网页中输入将文件加载到内存的指令 2.点击运行按钮	Action 界面会显示此指令运行成功	与预期效果一致
		验证将文件的所有文件块移动到 ssd 是否成功	1.在网页中输入将文件的所有文件块移动到 ssd 的指令 2.点击运行按钮	Actions 界面会显示此指令运行成功	与预期效果一致
		验证将文件的一个文件块移动到 ssd 是否成功	1.在网页中输入将文件的一个文件块移动到 ssd 的指令 2.点击运行按钮	Actions 界面会显示此指令运行成功	与预期效果一致

续表 5-3 测试用例表

		验证将文件的所有文件块移动到普通硬盘是否成功	1.在网页中输入将文件的所有文件块移动到普通硬盘的指令 2.点击运行按钮	Actions 界面会显示此指令运行成功	与预期效果一致
		验证将文件的所有文件块归档是否成功	1.在网页中输入将文件的所有文件块归档的指令 2.点击运行按钮	Actions 界面会显示此指令运行成功	与预期效果一致
	文件备份	验证将文件备份到另一个 Hadoop 集群是否成功	1.在网页中输入将文件备份到另一个 Hadoop 集群的指令 2.点击运行按钮	Actions 界面会显示此指令运行成功	与预期效果一致
		验证将文件备份到 WAS S3 是否成功	1.在网页中输入将文件备份到 WAS S3 的指令 2.点击运行按钮	Actions 界面会显示此指令运行成功	与预期效果一致
Rule 指令模块	批量删除文件	验证 Rule 指令能否被正确解析并正确运行	1.在网页中输入删除所有以.pdf 为结尾而且文件大小小于 5M 的文件的指令 2.点击运行按钮 3.激活此 Rule 指令	在 Rule 结果展示界面会看到其会产生多个 Action 任务, 并且所有.pdf 为结尾而且文件大小小于 5M 的文件会被删除	与预期效果一致

续表 5-3 测试用例表

	定时移动热文件到 ssd	验证 Rule 指令能否可以定时运行	1.在网页中输入每天将最近一天访问次数超过 5 次的文件移动到 ssd 的指令 2.点击运行按钮 3.激活此 Rule 指令	在 Rule 结果展示界面会看到其会定时产生多个 Action 任务，并且每天会将最近一天访问次数超过 5 次的文件移动到 ssd	与预期效果一致
	暂停 Rule 命令	验证 Rule 指令能否暂停运行	点击 Rule 任务暂停按钮	Rule 任务会暂停执行	与预期效果一致
	删除 Rule 命令	验证 Rule 指令能否可以删除	点击 Rule 任务删除按钮	Rule 任务会删除	与预期效果一致

经过系统功能性测试，全部功能可以正常运行，无错误。

5.3 非功能性测试

在保证系统功能正常运行之外，系统还应该满足非功能性需求，比如可靠性，安全性，兼容性以及性能等需求。下面对这几方面进行详细的测试。

1) 易用性：系统的界面是非常简洁的，主页就是命令输入的界面，其他界面是运行结果显示的界面。用户使用非常方便。在显示任务结果时，成功的任务有绿色标记，失败的任务是红色标记，这也是易用性的体现。系统通过 Rule 和 Action 两种指令，使得用户可以非常方便的的操作集群。

2) 兼容性：系统兼容 Hadoop2.7.3 及其以上版本的集群，对各种浏览器的兼容性都非常好。

3) 安全性：系统本身运行在公司的局域网当中，外网不能访问，而且有 kerberos 认证，系统非常安全。

4) 性能需求: 为验证系统的性能如何, 会统计系统完成任务所需要的时间, 以及任务执行时 CPU 和内存的使用率。系统指令执行性能统计如下表所示。

表 5-4 指令执行性能统计表

功能	CPU 使用率	内存使用率	预期运行时间	真实运行时间
向集群写 10MB 大小的文件	11%	18%	100ms	86ms
向集群中写 1GB 大小的文件	14%	23%	8000ms	5642ms
移动 100MB 大小的文件到 SSD 上	17%	16%	300ms	197ms
整合 20 个 5MB 大小的文件到文件容器中	21%	25%	200ms	124ms
向另一集群备份 1GB 大小的文件	14%	22%	15000ms	13298ms
批量删除文件系统中小于 1GB 而且存在时间超过一天的文件	29%	34%	5000ms	3845ms

由表中可以看出, 系统功能运行所需的硬件资源并不大, 而且运行速度较快, 高于预期。下面测试系统各页面的响应速度。

表 5-5 页面响应测试表

页面	预期时间	实际所用时间
提交 Action 指令	1s	0.6s
提交 Rule 指令	2s	1s
查看 Action 结果	1s	0.5s
查看 Action 结果具体信息	1s	0.5s
激活 Rule 指令	2s	1.5s
暂停 Rule 指令	1s	0.5s
删除 Rule 指令	1s	0.8s
查看 Rule 任务结果信息	1s	0.9s

根据上表分析: 系统页面的响应时间满足用户使用需求, 无明显卡顿。

因系统可能会同时执行大量的运维指令，所以对系统进行压力测试。通过 Python 脚本程序提交大量的 Action 指令和 Rule 指令，看系统的性能如何。

表 5-6 压力测试表

功能	CPU 使用率	内存使用率	预期运行时间	真实运行时间
同时运行 2000 个 Action 指令	57%	47%	1000s	894s
同时运行 10000 个 Action 指令	90%	67%	5000s	4561s
同时运行 200 个 Rule 指令	67%	56%	1000s	765s
同时运行 2000 个 Rule 指令	97%	89%	10000s	8765s

通过压力测试，系统能够支持大量任务的同时运行，系统依旧稳定运行，而且任务运行时间也符合预期。

5) 可靠性：系统设计之初就非常重视系统的可靠性。为抗单节点故障，系统还设置了两个 Master Server。经测试当某台机器宕机之后，系统依旧可以正常运行。当数据库服务器宕机时，重启机器之后，经测试系统也可以正常运行。经测试，系统稳定运行 24 小时，无明显异常，而且硬件资源占用合理。

5.4 本章小结

本章主要介绍了系统的功能性测试和非功能性测试。功能性测试，实际测试过程当中会使用大量的测试用例，以求全覆盖系统所有功能，但是篇幅有限，只列出最重要的若干测试用例。非功能性测试主要针对系统的可靠性，安全性，兼容性以及性能，经测试，系统满足了所有的非功能性需求。

6 总结与展望

6.1 总结

本系统是针对业界使用 Hadoop 集群作为其数据存储以及数据分析时的实际使用状况和存在的弊端进行的改进。目标就是为简化集群的运维操作，增强 HDFS 的应用场景。经过需求分析，系统设计，系统实现，以及测试，系统基本达到了预期目标。但是系统随着被业界落地使用可能会出现其他问题，而且系统并不完美，后续也会进行改进。本课题的研究主要内容如下：

针对 HDFS 运维的弊端：不够高效的运维指令，对小文件支持不友好，冷热数据无差别存储，没有很好的容灾备份机制。系统将集群中的元数据和集群访问记录同步到 MySQL 数据库中，利用这些数据对 HDFS 进行改进，主要改进成果如下：

- 1) 设计更加高效的集群运维指令，简化运维操作。
- 2) 支持冷热文件的不同存储策略。系统利用存放在数据库中的集群访问记录，来判定文件的冷热程度，并将其移动到相应的存储介质上。
- 3) 优化小文件存储。将小文件合并到一个大文件当中，并在数据库中存储小文件在大文件的索引信息。
- 4) 支持多集群备份。用户可以指定某些文件备份到另一个集群或者云存储中，实现集群更好的容灾备份能力。
- 5) 更加直观的运维结果展示。系统将任务运行的状态，结果信息通过前端界面展示出来。

6.2 展望

本系统虽然为 Hadoop 集群的运维做了优化，也对 HDFS 的存储场景做了很多改进，但是仍然有很多不足之处。

- 1) 系统过于重视后端系统的设计，前端界面的美观程度以及易用性可以再进行优化。

2) 系统应该支持数据压缩以及小文件的压缩^[36]。

3) Hadoop 本身已经支持纠删码来提高存储效率, 但是只有较高的版本才支持此功能。系统应针对这一场景, 支持较低版本的 Hadoop 集群也能使用纠删码提高数据存储效率, 而且支持块级别的纠删码^[37,38]。

4) 现有的 Action 命令并不丰富, 以后的改进中可以增加更多的指令来提高运维效率。

5) 进一步开发各大流行计算框架在本系统上的功能^[39], 为 Hive, Spark^[40], HBase 以及各种深度学习框架提供底层的数据存储功能。

致 谢

转眼间，两年半的硕士生涯即将结束，在求学过程中，虽有波折，但收获良多，非常感谢那些帮助过我的人，正是有了他们的帮助，我才能面对挫折，收获成长，才对未来的工作和生活充满希望。

感谢我的导师吴涛教授，老师丰富扎实的计算机理论知识、独到的学术眼光、认真的工作态度以及诲人不倦的师者风范将让我受益终生。在论文编写期间，从文章立意，创新点挖掘到论文格式，段落章节无一不——对我进行详细的指导，不仅在学习过程中循循善诱，更对我的生活态度、职业理想指引方向。我衷心的感谢导师两年半来对我的教诲。

感谢华中科技大学软件学院的老师们，和其他工作人员。正是在他们的帮助下我才能收获知识，增长见识，完美的度过我的硕士生涯。

感谢我的父母，家人以及女朋友高艳，在读研期间，他们的支持必不可少，虽然没有在学习上给我指导，但是家人的温暖让我不惧挫折，永攀高峰。

最后，我要向参与本人论文答辩的各位软件学院的老师致以深深的感谢。

参考文献

- [1] 李港,刘玉程.Hadoop 的两大核心技术 HDFS 和 MapReduce.电子技术与软件工程,2018(7):180-181
- [2] 王凌晖,解云月,周美华.Hadoop 分布式存储架构的性能分析.现代电子技术,2018,41(18):92-95
- [3] 刘涌,裴春梅,韩伟,高震宇.关于 Hadoop 中 HDFS 的研究.电脑知识与技术,2018,14(1):7-8
- [4] 严锐,兰奎,邹学利.一个健强的 AKKA 和 Spark 支持的大数据结构设计策略.决策咨询,2017(1):19-22
- [5] 刘涌,韩伟,赵静雅.关于 Hadoop 中 I/O 的研究.电脑知识与技术,2018,14(15):239-242
- [6] 李锋,贾茂想,涂如男.基于 Hadoop 的企业知识管理系统.计算机系统应用,2018,27(8):63-69
- [7] Uthayanath Suthakar, Luca Magnoni, David Ryan Smith, Akram Khan, Julia Andreeva. An efficient strategy for the collection and storage of large volumes of data for computation. Journal of Big Data, 2016, 3(1): 23-25
- [8] Wei Zhou, Dan Feng, Zhipeng Tan, Yingfei Zheng. Improving big data storage performance in hybrid environment. Journal of Computational Science, 2018, 26: 4-20
- [9] Mohd Rehan Ghazi, Durgaprasad Gangodkar. Hadoop, MapReduce and HDFS: A Developers Perspective. Procedia Computer Science, 2015, 48: 12-16
- [10] B.Saraladevi, N.Pazhaniraja, P.Victor Paul, M.S. Saleem Basha, P. Dhavachelvan. Big Data and Hadoop-a Study in Security Perspective. Procedia Computer Science, 2015, 50: 27-45
- [11] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung. The Google file system. ACM SIGOPS Operating Systems Review, 2003, 37(5): 10-23

- [12] Kun Wang. MOSM:An Approach for Efficient Storing Massive Small Files on Hadoop. Proceedings of 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA 2017) ,2017:5-7
- [13] 何龙,陈晋川,杜小勇.一种面向 HDFS 的多层索引技术.软件学报,2017,28(3):502-513
- [14] 段隆振,洪新利,邱桃荣.基于 MapFile 的 HDFS 小文件存取优化.南昌大学学报(工科版),2017,39(2):175-178
- [15] Xuhui Liu,Jizhong Han,Yunqin Zhong, Chengde Han, Xubin He.Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS. Cluster Computing and Workshops, 2009:23-35
- [16] Mariam Khader,Ali Hadi,Ghazi Al-Naymat. HDFS file operation fingerprints for forensic investigations. Digital Investigation,2018,24:12-45
- [17] Mostafa R. Kaseb,Mohamed H. Khafagy,Ihab A. Ali,ElSayed M. Saad. An improved technique for increasing availability in Big Data replication. Future Generation Computer Systems,2018:34-56
- [18] Kyoungsoo Bok,Hyunkyo Oh,Jongtae Lim,Yosop Pae,Hyoungtrak Choi,Byoungyup Lee,Jaesoo Yoo. An efficient distributed caching for accessing small files in HDFS. Cluster Computing,2017,20(4):20-34
- [19] 谭志远. 基于 Kylin 实现大数据多维分析. 广东通信技术,2018,38(8):48-51+58
- [20] 曹珍富,董晓蕾,周俊,沈佳辰,宁建廷,巩俊卿.大数据安全与隐私保护研究进展. 计算机研究与发展,2016,53(10):2137-2151
- [21] 陈左宁,王广益,胡苏太,韦海亮.大数据安全与自主可控.科学通报,2015,60(Z1):427-432
- [22] 冯登国,张敏,李昊.大数据安全与隐私保护.计算机学报,2014,37(1):246-258
- [23] Peng Shen. Research on Kerberos Technology Based on Hadoop Cluster Security.AEIC Academic Exchange Information Centre(China):International Conference on Humanities and Social Science Research,2018:6-10

- [24] 毕淏,程晓荣.Kerberos 认证协议分析与研究.电脑知识与技术,2017,13(27):37-38+59
- [25] 洪波,曹子建.基于 Hadoop 的分布式入侵检测系统设计与实现.西安工业大学学报,2018,38(4):390-395+407
- [26] 冯登国,张敏,李昊.大数据安全与隐私保护.计算机学报,2014,37(1):246-258
- [27] 陈左宁,王广益,胡苏太,韦海亮.大数据安全与自主可控.科学通报,2015,60(Z1):427-432
- [28] 张志,胡志勇.RESTful 架构在 Web Service 中的应用.自动化技术与应用,2018,37(10):33-37
- [29] 李斐.基于 Akka 的分布式集群运维系统设计与实现.东南大学,2017
- [30] 韦美雁,段华斌,周新林.大数据环境下的 MySQL 优化技术探讨.现代计算机(专业版),2018(30):68-72
- [31] 刘欢杰,魏静敏.关于 JAVA 的数据库连接池的探讨.信息系统工程,2012(5):44+52
- [32] Yuanyuan Qiao,Zhenming Lei,Lun Yuan,GUO Min-jie.Offline traffic analysis system based on Hadoop.The Journal of China Universities of Posts and Telecommunications,2013,20(5):97-103
- [33] V. O. Larin,O. V. Bantysh,O. V. Galkin,O. I. Provotar. The Domain-Specific Language Strumok for Describing Actor-Oriented Systems with Shared Memory. Cybernetics and Systems Analysis,2018,54(5):28-36
- [34] 胡博,陈桓,张良杰,牟建伟,戴广立,马于涛.一种跨 HDFS 集群的文件资源调度机制.计算机学报,2017,40(9):2093-2110
- [35] 刘景云.搭建 HDFS 分布式文件系统.网络安全和信息化,2018(2):100-103
- [36] 张航,马军.大数据平台数据压缩比和压缩效率研究.科技经济导刊,2018,26(17):33-50
- [37] 范涛,刘晓燕.基于纠删码的 HDFS 存储调度技术.软件,2018,39(2):173-178
- [38] 杨莲,马磊,吕传爱,李焱,尚明,佟峰.基于 RS 纠删码下 HDFS 云存储动态副本策

略的思考.科技创新与应用,2018(24):38-39

- [39] Wissem Inoubli,Sabeur Aridhi,Haithem Mezni,Mondher Maddouri,Engelbert Mephu Nguifo. An experimental survey on big data frameworks. Future Generation Computer Systems,2018,86:34-45
- [40] Jorge Ángel González Ordiano,Andreas Bartschat,Nicole Ludwig,Eric Braun,Simon Waczowicz,Nicolas Renkamp,Nico Peter,Clemens Döpmeier,Ralf Mikut,Veit Hagenmeyer. Concept and benchmark results for Big Data energy forecasting based on Apache Spark. Journal of Big Data,2018,5(1):5-45