一种跨 HDFS 集群的文件资源调度机制

胡 博[1),2)] 陈 桓[2)] 张良杰[1),2)] 牟建伟[1)] 戴广立[3)] 马于涛[4)]

[1)](金蝶软件(中国)有限公司大数据云平台部 广东 深圳 518057)

[2)](金蝶软件(中国)有限公司国家企业互联网服务支撑软件工程技术研究中心 广东 深圳 518057)

[3)](东南大学计算机科学与工程学院 南京 211189)

[4)](武汉大学软件工程国家重点实验室 武汉 430072)

摘 要 集群文件系统作为一种典型的分布式文件系统类型,通过集群内多个节点的协同,消除了单点故障以及性能瓶颈问题,实现了高可用、高性能以及动态负载均衡,并且具有较高的可扩展性,因此常作为实现和提供云存储服务的关键技术之一.该文针对 HDFS 集群主要局限于同一数据中心内部部署且可扩展性受限的问题,提出一种跨数据中心集群部署的文件资源调度机制和金蝶分布式文件服务 KDFS,通过分布式架构再设计,支持多个 HDFS 集群动态组网协同工作;通过引入文件资源池,屏蔽了不同集群之间的文件差异性,能够面向多应用提供透明服务;通过引入弹性存储与最优存储策略,确保集群资源安全冗余与就近服务的同时提升了集群的存储效率.实验和实践证明,跨 HDFS 集群的文件资源调度机制不但解决了 HDFS 集群可扩展性受限的问题,同时通过跨数据中心部署,实现了集群文件异地冗余灾备、跨数据中心负载均衡以及文件就近存取服务,有效地提高了应用使用 KDFS 存储服务的体验.

关键词 集群文件系统;分布式文件系统;HDFS;文件资源调度;云存储

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2017.02093

A Document Resource Scheduling Mechanism for HDFS Cluster Across Multiple Data Centers

HU Bo[1),2)] CHEN Huan[2)] ZHANG Liang-Jie[1),2)] MOU Jian-Wei[1)] DAI Guang-Li[3)] MA Yu-Tao[4)]

[1)](*Department of Kingdee Big Data & Cloud, Kingdee Software (China) Co., Ltd., Shenzhen, Guangdong* 518057)

[2)](*National Engineering Research Center of Supporting Software for Enterprise Internet Services, Kingdee Software (China) Co., Ltd., Shenzhen, Guangdong* 518057)

[3)](*School of Computer Science and Engineering, Southeast University, Nanjing* 211189)

[4)](*State Key Laboratory of Software Engineering, Wuhan University, Wuhan* 430072)

Abstract As a typical type of distributed file system, clustered file system, by coordinating several nodes within a cluster, reduces single-point failures and eliminates performance bottlenecks, succeeding in achieving high availability, high performance and dynamic load balance, as well as a relatively high scalability. As a result, it is one of the key technologies used for implementing and providing cloud storage services. This paper improves the limitation that HDFS can often be deployed into only one data center and its limited scalability and then proposes a document resource scheduling mechanism that allows to be deployed across data centers, and proposes Kingdee Distributed

收稿日期:2015-11-28;在线出版日期:2016-09-29.本课题得到国家"九七三"重点基础研究发展规划项目基金(2016YFB1000800)、国家云计算示范工程(发改办高技[2011]2448 号)、国家发改委大数据示范工程(发改办高技[2014]648 号)、广东省信息产业发展专项(粤经信电软[2015]141 号)、广东省领军人才计划([2012]342 号)、深圳市技术开发项目(FWY-CX20140310010238)及广东省领军人才计划(粤人才[2012]342 号)资助.**胡 博**,男,1983 年生,博士,研究员,中国计算机学会(CCF)会员,主要研究领域为云计算、大数据.E-mail:bob_hu@kingdee.com. **陈 桓**(通信作者),男,1984 年生,博士,中国计算机学会(CCF)会员,主要研究方向为云计算、大数据.E-mail:huan_chen@kingdee.com. **张良杰**,男,1969 年生,博士,IEEE 会士,中国计算机学会(CCF)会员,主要研究方向为服务计算. **牟建伟**,男,1988 年生,学士,工程师,主要研究方向为云存储. **戴广立**,男,1995 年生,学士,工程师,主要研究方向为机器学习. **马于涛**,男,1980 年生,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为服务计算.

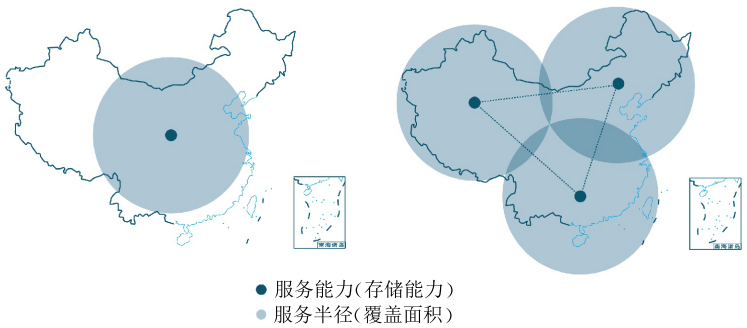
File System(KDFS). By redesigning distributed architecture, it supports dynamic network reconfiguring of multiple HDFS clusters, enabling work coordination within the cluster; meanwhile, this paper introduces file resources pool, shielding the differences among clusters, providing transparent services to multiple applications. This paper also introduces elastic storage and optimal storage strategies, ensuring safety redundancy of cluster resources, near-site services, as well as storage efficiency. In experiments and practices, this inter-HDFS clusters document resource scheduling mechanism not only solved the scalability problem of HDFS, but also succeeded in achieving cluster files offsite disaster recovery, load balancing across data centers, and near-site file storage services, by the way of deploying the system across data centers. These proposed approaches effectively improve the user experience with KDFS storage services.

Keywords clustered document system; distributed file system; HDFS; document resource scheduling; cloud storage

1 引 言

集群文件系统是指运行在计算机集群之上,将相关存储资源整合、虚拟化并对外开放接口,提供海量大规模文件存储和访问服务的一种分布式文件系统(Distributed File System,DFS)[1-2].典型的集群文件系统主要有谷歌文件系统(Google File System,GFS)[3]以及 Apache 社区针对 GFS 推出的开源工程 Hadoop 分布式文件系统(Hadoop Distributed File System,HDFS)[4]等,其基本原理是利用几十、上百台甚至更多的计算机作为节点相互连接组成一个计算机网络集群,通常以中心网络模型提供服务.在这个集群中,绝大多数节点作为存储节点负责存储数据,而一些节点则作为跟踪节点负责管理存储节点之间的协同.

较之于传统的计算机文件系统,集群文件系统通过多个节点之间的协同工作,消除了单点故障以及性能瓶颈问题,多数还利用廉价的计算机基础设施实现了高性能、高可用性以及动态负载均衡,并且具有较高的可扩展性.因此,集群文件系统往往被视为云计算的基础设施之一,在云计算、云存储等应用中发挥关键作用.然而如图 1 所示,作为目前最流行的分布式文件系统之一的 HDFS,由于主要支持将所有的节点部署在同一数据中心集群内部,难以实现跨数据中心的部署[5-6],一定程度上制约了 HDFS 的最大服务半径;同时,由于 HDFS 跟踪节点 name node 难扩展性也阻碍了集群无限扩充的可能[7],使其服务能力受限.因此当业务发展到一定阶段时,当前的系统已经无法满足新需求.



(a) 单集群部署服务能力与服务半径 (b) 跨数据中心集群服务能力与服务半径

图 1 单集群及跨数据中心集群 DFS 服务能力与服务半径

本文拟针对 HDFS 集群主要局限于同一数据中心内部部署且可扩展性受限的问题,提出一种跨数据中心集群部署的文件资源调度机制及金蝶分布式文件服务(Kingdee Distributed File System,KDFS),通过分布式架构再设计,支持多个 HDFS 集群跨数据中心横向扩展、动态组网和透明访问;并通过对文件在跨数据中心存储节点的最优分布研究,实现文件在 KDFS 内的最优存储和就近存取.

本文第 2 节主要分析 HDFS 集群在可扩展性和面向多应用提供透明服务方面的问题以及针对该问题的相关研究工作，并针对相关工作研究之不足指出本文的主要创新方向；第 3 节给出支持跨数据中心集群部署的新型分布式文件系统 KDFS 定义；第 4 节在第 3 节基础之上设计 KDFS 架构及其分布式资源管理机制；第 5 节重点研究研究分布式资源管理机制中的动态组网策略，解决了新文件上传以及 KDFS 存储节点网络发生改变后文件如何最优分布的问题；第 6 节则在生产环境中实现了 KDFS，并通过实验验证跨数据中心集群部署的 KDFS 在存储体验方面整体性能要优于单一数据中心部署的 HDFS 集群，以及证明动态组网机制在 KDFS 内部行之有效. 第 7 节是本文的结论及后续工作展望.

2　问题及相关工作

存储是云计算、大数据的核心基础设施之一，负责对应用系统提供文档、业务数据、运行时环境等不同类型数据资源的永久存储、短期存储或缓存等功能. 它的可扩展性、可分布性、可用性、安全性以及性能和容量等都直接影响应用系统的服务质量，因而各大厂商都尝试不断优化存储这个核心要素.

众所周知，谷歌的三大基石是 GFS、Bigtable[8] 和 MapReduce[9]，这里含有存储特性的组件 GFS 与 Bigtable 为 MapReduce 提供关键资源供给支撑. 其中，GFS 处于其他组件的最底层，是一个典型的分布式文件系统，用于大型的、大量的、分布式的文件访问. 它提供容错特性，并可以运行于廉价存储之上，提供非性能密集型文件访问. HDFS 作为 GFS 的开源实现，继承了 GFS 绝大部分特性. 在 Hadoop 0.23 版本之前的 HDFS 主要包括两层结构：命名空间层(namespace)和块存储层(block storage)，前者用于管理目录、文件和数据库，支持常见的文件系统操作如文件的创建、修改、删除等，主要依赖 name node 完成. 后者则关注数据块的相关操作，如创建数据库、删除数据库等，其中 name node 承担数据库的管理职能，如维护集群中存储节点的基本关系，管理数据块副本的复制和存放. 而 data node 主要负责数据块的存储. HDFS 的出现，为广大厂商提供了可重用的关键存储模块，极大地降低了开发成本，但依然存在如下两方面问题：

(1) 集群的可扩展性问题

大规模小文件存储一直是 HDFS 为人诟病的弊端之一，但问题的症结实际并非在于 HDFS 不适合小文件存储，而是由于 HDFS 的扩展能力有限导致其存储小文件的成本要远远大于大文件的存储成本. 众所周知，文件在 HDFS 的存储节点 data node 中是按 Block 来存储并被表示成固定占用 150 bytes 的对象注册在跟踪节点 name node 的 namespace 当中的. 由于 HDFS 规定每个 Block 都有一个大小限制(如 64 M)，因此当 HDFS 去存储一个小于 Block 规定大小的文件的时，也会独占一个 Block 造成空间浪费. 而 HDFS 架构又只允许整个集群只有一个 namespace 存在并受 name node 内存的限制，所以一个 HDFS 集群中最大能够允许多少个 Block 存在或者说一个 HDFS 可能存储的最大空间容量的决定因素是 name node 中的 namespace 而并非 data node. 因此，在 HDFS 横向扩展能力受限、空间大小有限的制约下，大规模小文件存储就显得不是那么经济，从而让人们认为 HDFS 不适合小文件存储，实际上 HDFS 在大规模大文件存储依然会遇到存储容量的瓶颈问题，亟待通过 HDFS 扩展来解决.

针对小文件的存储优化工作，Hadoop 自带工具 Hadoop Archive 和 Sequence file 等提供了相应解决方案，也有文献[10-13]研究通过改进 HDFS 的存储特性，能够在一定程度上提高大量小文件的存取效率，但没有本质上解决 HDFS 集群扩展的根本问题. HDFS Federation[14] 是 Hadoop 0.23 版本及以后为了解决 HDFS 单点故障而提出的 namespace 水平扩展方案，该方案允许 HDFS 创建多个 namespace 用以提高集群的扩展性和隔离性. 阿里巴巴云梯[15-16] 在 HDFS Federation 基础之上通过实现 namespace 跨机房切分从而实现了数据的跨机房分布. 以上两个方法主要是通过实现对多个 name space 的支持来解决同一 HDFS 超大规模集群扩展的问题. 而本文则希望通过多 HDFS 集群协同工作实现多 namespace 支持来达到存储容量扩展的目标，并通过跨数据中心集群部署与协同服务来给全国不同区域用户提供就近的文件存取服务，既能提升系统的服务能力，同时也能够加大系统的服务半径. 当然，为此也需要解决由于 HDFS 跨数据中心集群部署带来的系列相关问题，如文件跨集群最优分布问题等.

(2) 多应用以及透明服务的问题

存储的目标是通过应用面向用户提供服务. 同时服务于多个应用能够最大化提升存储效率并降低存储成本，同时数据的共享还能够给用户带来极致

的体验.在面向服务的软件工程中,资源的服务化和服务的通用化要求资源成为一种服务后应该具备面向多个应用提供服务的能力[17].就存储服务而言,应用即租户,且 HDFS 在多租户特性方面有所欠缺:它不支持多用户同时操作一个文件[18],这使得底层文件共享成为难点.它也不支持文件的任意位置修改,文件只能通过替换或追加的方式来修改.这两方面特性给 HDFS 的应用特别是面向多应用提供通用存储服务带来了极大的不便,因而各大厂商都尝试通过各种方法弥补不足[19-20].典型的做法是通过在 HDFS 文件逻辑层上再构建一层文件操作与属性层,使得用户可以通过该层实现对文件的多租户操作.但随着新的一层多租户操作层的构建,原本很多 HDFS 特性无法自然继承,需要通过复杂的改进开发来完善[18].本文拟通过引入文件资源池实现对物理文件的逻辑管理但不破坏和影响 HDFS 集群本身的元数据操作,文件资源池通过对多租户数据架构的支持,实现集群能够面向多个租户(应用)提供服务.另外,HDFS 由于在一定层面上遵从 POSTFIX[21]标准,所以也可以较为轻松地像 Linux 一样操作文件.但实现跨 HDFS 集群的部署无疑会带来一些新的问题,如文件上传到哪个集群,文件从哪个集群获取,以及文件如何从一个集群迁移到另外一个集群等都会造成使用的复杂性,如何屏蔽由此带来的复杂性给应用带来透明的体验也是本文拟解决的问题之一.

综上,本文拟针对分布式文件系统可扩展性问题展开研究,基于当前流行的分布式文件系统 HDFS,通过分布式架构再设计和文件存储和传输机制的优化改进等解决系统的跨数据中心扩展、文件最优分布等关键难点,实现数据节点自治服务、动态组网、最优存储、就近存取和透明访问,并设计相关系统.

3 KDFS 定义

KDFS 是一套全新的分布式文件系统,基于传统文件系统的本地文件存储能力,实现了跨数据中心集群的存储能力扩展及分布式文件资源调度.目前的 KDFS 主要兼容 HDFS 等系统,拟通过分布式架构再设计和文件传输和存储机制的优化改进等解决 HDFS 难以跨数据中心扩展的问题.KDFS 采取经典的中心网络模式,其中存储节点称为 K-NODE,而跟踪节点则随之称为 K-SERVER.

定义 1. K-NODE.为 KDFS 存储节点,每一个 K-NODE 实际上是对部署在一个特定数据中心的 HDFS 集群的封装,能够完整地提供数据上传、下载、管理以及其他专业技术服务(如图片缩略等),并在 K-SERVER 监管和指引下具备独立面向应用提供服务的能力,同时兼具多个 K-NODE 共同组网协同面向应用提供服务的能力.形式上可表达为

$$\text{HDFS} \wedge \text{HDFS-Agent} \Rightarrow \text{K-NODE}.$$

其中,HDFS 集群负责命名空间管理、文件存储与文件操作,集群功能和最大规模与传统 HDFS 集群无异.HDFS-Agent 作为 HDFS 集群的代理,将 HDFS 中的 name node 和 data node 对外完全透明化,并以一组统一的服务接口与 K-SERVER 交互并在其调度下直接面向上层应用提供文件存储服务.

定义 2. K-SERVER.为 KDFS 跟踪节点,用于集中监控和调度 KDFS 中的存储节点 K-NODE.以注册监听模式监控 K-NODE 健康状态,并实时对所有健康的 K-NODE 进行动态组网,形成 K-NODE 网络以协同面向应用提供服务;维护 KDFS 全局唯一的文件资源池,制定物理文件在 K-NODE 上的存储策略以及在不同 K-NODE 之间的同步策略;同时面向上层应用提供文件查询与寻址服务,引导应用从最适合的 K-NODE 存储和获取文件.形式上:

$$\text{KDFS-Scheduler} \wedge \text{FRP} \Rightarrow \text{K-SERVER}.$$

其中,KDFS-Scheduler 用于接受 K-NODE 注册,监督 K-NODE 监控状态,当有多个 K-NODE 注册时负责对多个 K-NODE 进行动态组网从而确保不同的存储节点之间能够协同工作 FRP(File Resource Pool,文件资源池)负责维护一个逻辑的全局文件清单以及该清单到文件实际在 K-NODE 中存储物理位置的映射,并对外屏蔽底层物理存储的差异性,面向上层提供一致的文件索引服务.

图 2 描述了 KDFS 的网络拓扑结构.其中,K-NODE 在 K-SERVER 监督和指导下进行组网并通过相互协同工作.在整个网络中,至少包括 1 个 K-NODE 节点以及 1 个 K-SERVER 节点,形式上:

$$\text{K-SERVER} \wedge \text{K-NODE} \Rightarrow \text{KDFS}.$$

而在实际的部署和运营过程中,为了保证系统的存储能力和服务能力,往往会在不同的数据中心部署多个 K-NODE,而为了保证 K-SERVER 可用性,这里引入 K-SERVER-HA 来去单点化并通过集群部署实现跟踪节点的动态负载均衡.

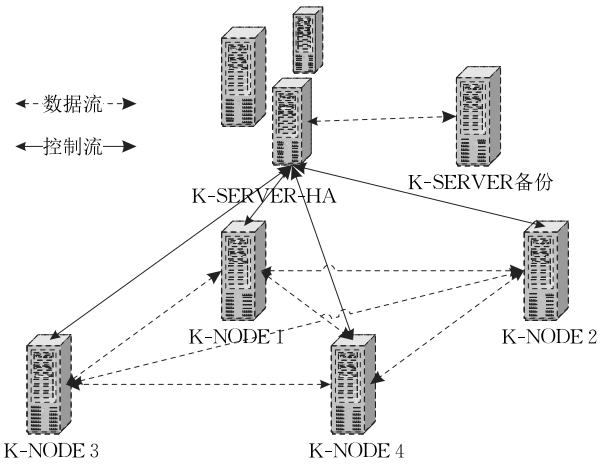


图 2　K-NODE 网络拓扑结构

定义 3.　K-SERVER-HA. 是 K-SERVER 的动态高可用节点集群，支持在线提供 K-SERVER 的失效自动切换以及高负载情况下的动态负载均衡. K-SERVER-HA 包括主干节点 $\text{K-SERVER}^{\text{Main}}$ 和备份节点 $\text{K-SERVER}^{\text{Backup}}$，一旦主干节点失效将从备份节点选举产生：

$$\text{K-SERVER}^{\text{A}} \Leftrightarrow \text{K-SERVER}^{\text{Main}} \oplus \text{K-SERVER}^{\text{Backup}}.$$

综上，给出 KDFS 的完整定义.

定义 4.　KDFS. 一种跨数据中心集群部署的分布式文件系统，兼容并支持 HDFS 等分布式文件系统跨数据中心多集群部署，并实现跨数据中心集群的资源调度与协同服务. 形式可定义为

$$\sum_{k=1}^{n} \text{K-NODE}_k \wedge \text{K-SERVER}^{\text{A}} \Leftrightarrow \text{KDFS}.$$

$\text{K-SERVER}^{\text{A}}$ 为 K-SERVER-HA 的高可用在线服务节点. KDFS 为 HDFS 的超集，覆盖 HDFS 所有功能，并扩展了一些生产上所必须的特性. 当 KDFS 为前端应用提供服务时：

$$\text{K-SERVER}^{\text{A}} \Leftrightarrow \text{K-SERVER}^{\text{Main}} \oplus \text{K-SERVER}^{\text{Backup}}.$$

即面向应用时，$\text{K-SERVER}^{\text{Main}}$ 或 $\text{K-SERVER}^{\text{Backup}}$ 仅有一个在线并提供服务. 对于 KDFS 系统自身，其备份节点 $\text{K-SERVER}^{\text{Backup}}$ 模型为

$$\text{RML} \wedge \text{K-SERVER}^{\text{Backup}} \Leftrightarrow \text{K-SERVER}^{\text{Main}},$$

$$\text{RML} \Leftrightarrow 1.$$

$\text{K-SERVER}^{\text{Backup}}$ 是 $\text{K-SERVER}^{\text{Main}}$ 的实时逻辑镜像. RML(Resource Mutex Lock，资源互斥锁)保证在同一时刻同一资源只能被某一实体占用，此处 RML 保证了文件资源在同一时刻只能被 $\text{K-SERVER}^{\text{Main}}$ 或者 $\text{K-SERVER}^{\text{Backup}}$ 两者之一控制和使用. 正常情况下，两者完全一致且可互换. 当局部功能失效，则 $\text{K-SERVER}^{\text{A}}$ 服务能力降级为两者服务的交集. 需要注意的是，此时刻仍然只有一个 $\text{K-SERVER}^{\text{A}}$ 提供服务，由 RML 保证. 例如，$\text{K-SERVER}^{\text{Main}}$ 下载功能失效，上传功能正常，则 $\text{K-SERVER}^{\text{A}}$ 的下载功能短暂失效. 此时，$\text{K-SERVER}^{\text{Backup}}$ 会在极短的时间内，自动替换 $\text{K-SERVER}^{\text{Main}}$，恢复 $\text{K-SERVER}^{\text{A}}$ 完整的服务能力.

表 1 描述 K-SERVER-HA 各组件之间的状态变换关系. 根据 $\text{K-SERVER}^{\text{A}}$ 形式化定义，当 $\text{K-SERVER}^{\text{Main}}$ 失效时有如下感知及修复步骤：

主节点失效感知：$\text{K-SERVER}^{\text{Main}}=0$；

互斥锁感知：$\text{K-SERVER}^{\text{Main}}=0=\text{RML}\wedge\text{K-SERVER}^{\text{Backup}}=1\wedge\text{K-SERVER}^{\text{Backup}}$，此时 K-SERVER-HA 节点的可用性取决于 $\text{K-SERVER}^{\text{Backup}}$，当 $\text{K-SERVER}^{\text{Backup}}$ 可用(即 $\text{K-SERVER}^{\text{Backup}}$ 为 1)，与实际 $\text{K-SERVER}^{\text{Main}}=0$ 矛盾，必将要求 RML 为 0；

主备切换：要求与约束 $\text{RML}\Leftrightarrow 1$ 矛盾促使主备切换 $\text{K-SERVER}^{\text{Main}}=\text{K-SERVER}^{\text{Backup}}$.

表 1　K-SERVER 状态转换对照表

$\text{K-SERVER}^{\text{Main}}$	$\text{K-SERVER}^{\text{Backup}}$	$\text{K-SERVER}^{\text{A}}$
正常	待机	正常
正常	失效	正常
失效	正常	正常
失效后修复待机	正常	正常
失效	失效	失效

K-SERVER-HA 实现机制与普通高可用应用服务类似，使用数据库主从热备技术[22]、高可用自动切换技术[23]、负载均衡技术[24-25]、虚拟网络技术实现，主要用于避免单点故障.

4　集群架构与分布式资源管理

4.1　架　构

根据 KDFS 定义，设计集群架构如图 3 所示，共分 4 个层次，分别是 K-NODE 网络层(物理存储层)、K-SERVER-HA 层(逻辑存储层)、KDFS 接口层以及应用层.

顾名思义，K-NODE 网络层由多个 K-NODE 节点连接组网而成，在整个 KDFS 中实际负责存储海量文件资源，因此又称之为物理存储层. 在这一个层次中，每个 K-NODE 节点都是对一个独立 HDFS 集群的封装(实际配以其他的 Agent 亦能够封装其他类型文件系统用以对特定类型的文件存储，如 GridFS 等)，并单独部署于一个特定的数据中心，因此单个 K-NODE 节点的存储容量和部署地点

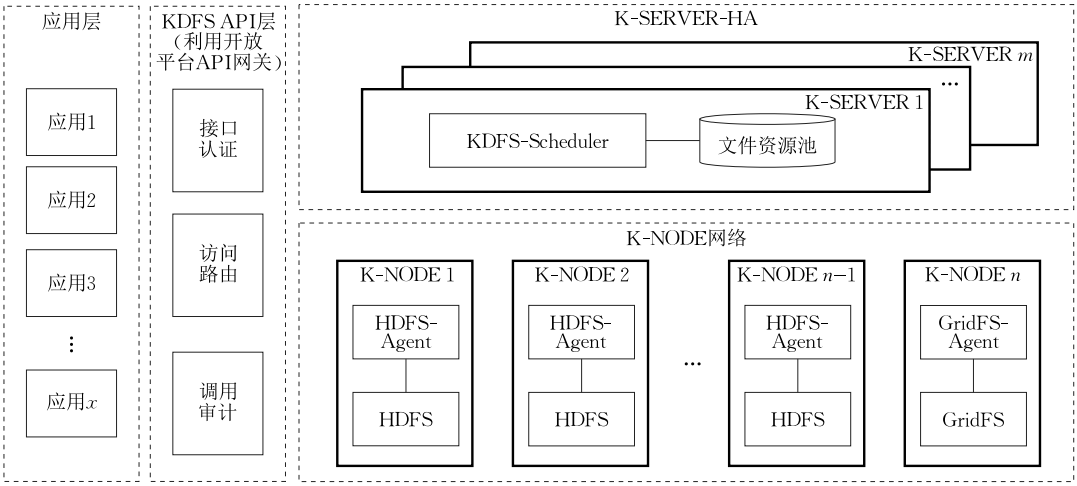


图 3 KDFS 架构

直接决定了它在 K-NODE 网络中的存储能力与服务半径. 一般而言,一个 K-NODE 网络通常有 m 个 K-NODE 节点组成,其中 $m\geq 1$. 这些存储节点能够按照 K-SERVER 要求组成一个存储网络,以联合覆盖半径最大为目标而通常会选择最优的网络空间位置进行部署,如就服务国内用户而言,K-NODE 网络部署最优通常选择部署在北京、广州、上海、成都或西安等网络骨干节点所在地,以满足华东、华北、华南、西南、西北地区的服务覆盖. 当有新 K-NODE 节点并网加入到 K-NODE 网络或已并网 K-NODE 节点由于宕机或其他故障无法提供服务的时候,K-NODE 网络会根据实际能够提供服务的 K-NODE 节点个数在 K-SERVER 指导下重新进行组网,又称动态组网. 在 K-NODE 网络中,文件及其多个副本分布式存储在不同的 K-NODE 之中,具体该文件需要存储多少个副本以及这些副本如何分布式存储由 K-SERVER 统一调度.

K-SERVER 负责指挥 K-NODE 动态组网以及文件及其副本的分布式存储,其关键在于 KDFS-Scheduler 和文件资源池两个核心组件. KDFS-Scheduler 用于监听各个 K-NODE 节点的健康状况,响应上层应用请求并告之其文件实际需要存放或获取的 K-NODE 节点,指挥文件及其副本在不同 K-NODE 之间的迁移. 文件资源池则为 KDFS-Scheduler 提供了文件资源状态的数据库支持,负责统一管理 KDFS 所有文件及其在不同 K-NODE 之间的存储和迁移信息,它遵循多租户数据架构,支持不同租户的数据逻辑或物理上隔离管理. 由于K-SERVER 的可用性和稳定性非常关键. H-SERVER-HA 层负责维护一个 K-SERVER 的高可用集群,集群包括 n 个 K-SERVER 节点,其中 $n\geq 2$,至少包括一个 K-SERVER 主干节点以及备份节点. K-SERVER 主干节点及其备份节点可以采取多种高可用策略,如 Stand By 策略或 Read Only 策略. 前者备份节点平时闲置,当且仅当主干节点确定失效的时候才被选举担当新的主干节点;后者平时备份节点则需要分担主干节点信息读取的请求负荷,当且仅当主干节点确定失效后才被选举出来担当新的主管节点. 无论 K-SERVER 节点个数是多少,对 K-NODE 均是透明的. K-SERVER-HA 集群服务发基于 etcd 实现,主干节点选取机制见文献[26-27],本文不再赘述.

KDFS 架构决定 K-SERVER 只用于管理 K-NODE 网络以及决定文件具体存取的节点,并对应用屏蔽底层架构的复杂性,而应用从 K-SERVER 获取实际存取的存储节点后实际的存取工作则直接在应用和指定的 K-NODE 之间完成. 因此,无论是 K-SERVER 还是 K-NODE 都必须面向应用暴露接口,支持应用通过接口完成文件存取所需的必要操作. KDFS 通过 API 层统一对外开放接口,业务涵盖存储节点选取、存储空间开通与管理、文件(分块)上传、文件(断点)下载、文件的版本化以及文件的二次加工(如快照和缩略图)等方面. 在接口技术支持方面,KDFS API 层借助“金蝶云开放平台”API 网关的能力,为接口的访问的合法性进行认证和授权,对可能分布式部署的 K-SERVER-HA 集群去路由选择正确的主干节点,以及对实时的访问进行监控和合规性审计,同时也可以用于统计应用对 KDFS 接口的调用次数、存放的文件数量以及空间使用的用量等情况. 在开放接口风格方面,KDFS 采取标准的 REST 架构风格封装 Web 服务,这样做的好处是轻量级的接口能够有效降低开发的复杂性同时提高

系统的分布式可伸缩性.

基于 KDFS 对 HDFS 的封装以及文件资源池对多租户架构的支持,使得 KDFS 天然具备跨数据中心部署的能力以及面向多应用提供服务的能力.任何应用通过 KDFS API 层申请成为服务消费者,都可以直接调用 KDFS 提供的接口将用户的文件存储在 KDFS 之中.

综上论述,KDFS 架构的特点在于:

(1)实现了跨 HDFS 集群的横向扩展.每一个存储节点 K-NODE 均是通过 HDFS-Agent 对一个 HDFS 集群的封装,单个 K-NODE 节点的最大存储容量视 HDFS 集群可存储的最大容量而定.由于 KDFS 最多支持 m 个 K-NODE 节点协同面向应用提供存储服务,且理论上 $m\geq 1$,尽管上限未知但确定能够成倍突破了单个 HDFS 集群能够存储最大数据的局限,实现了 HDFS 跨数据中心部署以及集群的扩展.

(2)实现了分布式协同存储与就近服务.当 $m>1$ 的时候,通过 K-SERVER 动态组网算法,能够让加入到 KDFS 集群中的 K-NODE 节点相互之间形成一个互联互通的网络,利用网络让文件得以在不同的存储节点之间同步达到最优分布,不但实现了文件资源跨存储节点的安全冗余备份,还支持将用户的文件存储在离用户最近的存储节点,从而实现就近服务.

(3)实现了多租户和透明服务.K-SERVER 文件资源池通过支持多租户数据架构,实现不同租户之间的数据逻辑隔离,支持 KDFS 同时面向多个应用提供存储服务.同时,存储节点 K-NODE 的选取由跟踪节点 K-SERVER 根据 KDFS 各存储节点状态以及与用户最佳的位置实时推荐,不需要应用选择,且每个 K-NODE 节点接口与操作均保持一致,降低了应用接入 KDFS 的难度,实现了透明化服务.

同时,提供了高可用保障.通过 K-SERVER-HA 以及服务发现和主干节点选举机制破解了 K-SERVER 可能存在的单点问题,实现了跟踪节点的高可用.同时备份用的 K-SERVER 节点同样能够提供文件资源池读取服务,能够分担跟踪节点访问压力实现负载均衡.

4.2　文件资源池

数据的一致性是分布式系统协同工作的关键问题之一.KDFS 解决不同 K-NODE 节点之间数据不一致问题的方案正是利用文件资源池作为基础,并通过弹性存储机制和文件传输机制实现了全局文件的统一管理.一方面,文件资源池作为逻辑存储屏蔽了底层存储节点中物理存储的差异性,对于应用来说首先不需要关心文件具体存储在那个 K-NODE 之上,较之于 P2P 网络无需泛洪式查找而只需要根据 K-SERVER 指导去具体的存储节点存取信息;另一方面,文件资源池也能够动态维护逻辑存储与物理存储的映射关系,并通过 KDFS-Scheduler 来指导应用应该将文件就近存储在哪个存储节点之上或者应该从那个存储节点就近下载,以及文件在已存储的存储节点与远端节点之间的同步或迁移.

图 4 描述逻辑文件与物理文件的映射关系,逻辑文件由 K-SERVER 上文件资源池维护,而物理文件则存储在 K-NODE 之上.对于逻辑文件而言,一个文件在全局范围内有且仅存一份,而一个逻辑文件对应的物理文件可能存在于不同节点的不同位置.图中文件资源池中的文件“钢铁是怎样炼成的.pdf”就被标识成文件编码为“12123.5”的资源并分别存储在 D1、D2 和 D5 三个存储节点中.

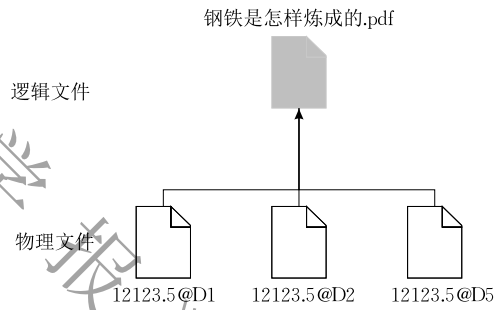


图 4　逻辑文件与物理资源的映射关系

极端情况下,一份物理文件最少被存储在一个存储节点上即可正常提供服务,但为了确保数据的安全,一份物理文件通常至少应该被冗余存储在两个不同的存储节点以防止其中某个节点由于宕机而无法提供服务.在 KDFS 中,一份文件会根据文件弹性存储策略被制作成多个副本并根据文件最优分布策略存储在多个合适节点中,更多的冗余能够避免多点故障发生减少停止服务几率的同时,也可以利用 KDFS 允许每个 K-NODE 能够单独面向用户提供服务的特性,让用户能够在就近的存储节点上存储或读取数据,提高了文件的存取效率以及用户体验.

4.3　弹性存储策略

当 K-NODE 网络中存储节点数目等于 2 的时候,每个 K-NODE 节点正好存储一个文件副本,从而能够在安全冗余的前提下实现每个副本都能够就

近面向 K-NODE 附件用户提供服务,这被认为是一种最经济的文件存储方案.但在实际生产过程中,2 个 K-NODE 的服务半径往往并不能够完全满足需求,会有更多的存储节点加入协同面向更多的用户服务,但如果在每个存储节点上都存放一个文件副本,这钟策略无疑能够给用户带来最好的体验,但如此之多的副本在 K-NODE 网络中传输和存储会给运营成本的控制带来挑战.

弹性存储的目标是为了在文件及其副本存储总数控制与经济性方面达到一个平衡,即既能够保证 K-NODE 网络中存放有足够多的文件副本以面向用户提供就近服务,同时能够将文件及其副本数控制到最少.

KDFS 弹性文件存储策略将文件及其副本数划分为两类:冷备份与热备份.注意的是,本文认为文件及其副本在 K-NODE 网络中平均分布,暂不考虑 K-NODE 与用户活动给文件物理存储造成的偏好影响.

定义 5. 文件冷备.是指文件存储在 K-NODE 中的基本副本,当存储节点数目为 N 时,文件的冷备数为 $\left[\frac{N-1}{2}\right]$.一旦存储文件冷备的 K-NODE 失效,文件冷备必须在其他节点重新建立.

定义 6. 文件热备.是指文件存储在 K-NODE 中的扩展副本或临时副本,当存储节点数目为 N 时,文件的热备数最大可以为 $\left[\frac{N+1}{2}\right]-\left[\frac{N-1}{2}\right]$.当 K-NODE 存储不足时可以删除文件热备.一旦存储文件热备的 K-NODE 失效,文件的热备亦无需考虑在其他节点重新建立.

图 5 描述了当 K-NODE 网络中存储节点数为 5 的情况下文件的分布情况.根据弹性存储策略,此时文件的冷备数最少为 2 个,而文件的热备数最多为 1 个.文件的冷备一般有一份存储在就近上传的节点上,而其他的冷备及其热备则平均分布在 K-NODE 网络其他存储节点中.

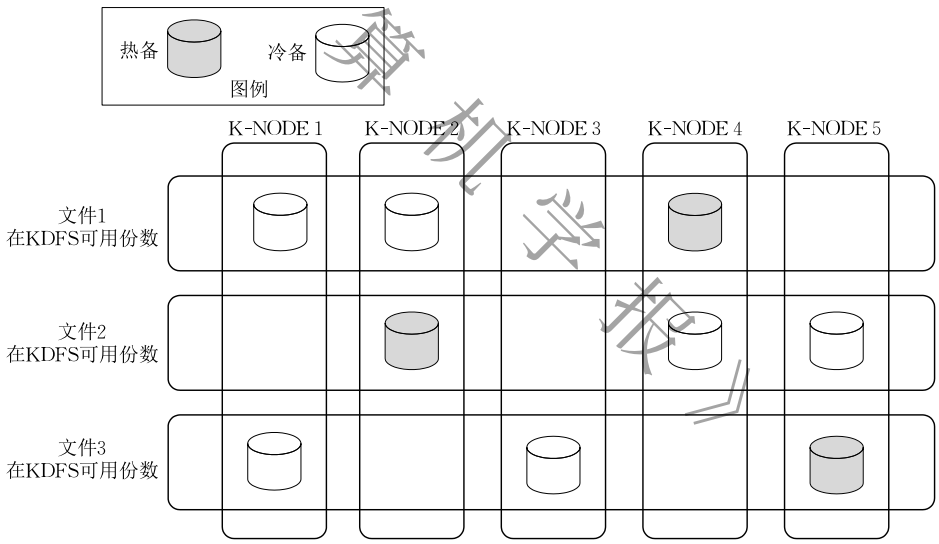


图 5 KDFS 弹性存储策略

4.4 文件上下载机制

KDFS 文件上下载机制的特点是透明:应用无需事先选择或指定文件即将上传到哪个 K-NODE,或是文件具体存储在哪个 K-NODE,以及文件在 K-NODE 之间是如何分布的,只需要在向 KDFS 发送文件上传或下载请求时向 K-SERVER 询问并将文件上传到 K-SERVER 实时推荐的 K-NODE 或从 K-SERVER 实时推荐的 K-NODE 下载.

4.4.1 询问上传

图 6 描述了应用将一个文件上传到 KDFS 并存储的流程.在这个流程中,应用在上传文件前首先向 K-SERVER 提出上传文件的请求,K-SERVER 接受请求后会根据所了解的 K-NODE 监控状态,并结合用户的网络空间位置,将 K-NODE 网络中负载最轻且离用户网络空间距离最近的 K-NODE[i]节点推荐给应用,并告知应用文件上传到 K-NODE[i]的具体地址;应用获取文件上传地址,便直接将文件通过 HTTP 方法 PUT 或 POST 到 K-NODE[i];当文件上传成功后,K-NODE[i]报告 K-SERVER,待 K-SERVER 维护完文件资源池后再通知用户文件上传成功.

询问上传流程的特点是:(1)即便同一个用户

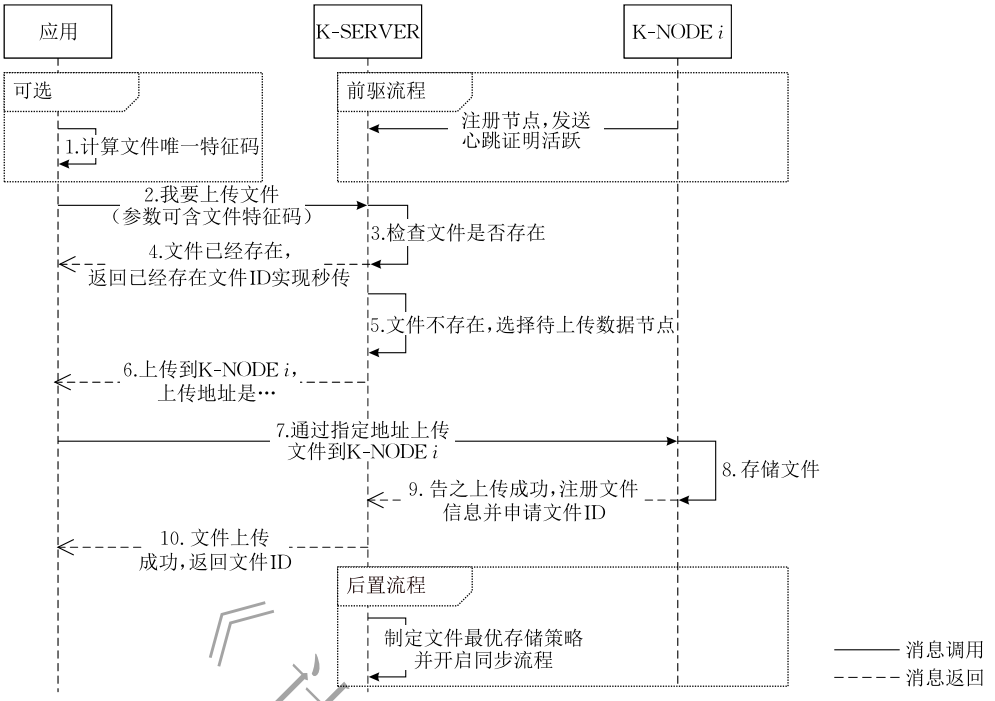


图 6　询问上传流程

在同一应用中上传文件，文件首次存储的 K-NODE 也会有所不同，K-SERVER 推荐 K-NODE 的原则是其网络空间位置离用户较近且健康状态良好，不会有太重的负载；(2) 支持文件的秒传，即如果应用事先根据约定的算法（如 MD5 算法）计算出待上传文件的唯一特征码后，一旦 K-SERVER 发现该文件存在于文件资源池中，即可直接返回 KDFS 中相同文件的 ID 到应用，实现文件秒传.

4.4.2　询问下载

图 7 描述了应用从 KDFS 下载一个文件到本地的流程. 与询问上传类似，应用首先向 K-SERVER 提出下载请求；K-SERVER 接受请求后会根据所了解的 K-NODE 监控状态，并结合用户的网络空间位置，将 K-NODE 网络中负载最轻且离用户网络空间距离最近的 K-NODE[i]节点推荐给应用，并告知应用文件下载的具体地址；应用获取文件下载地址，便直接将文件通过 HTTP 方法 GET 到本地即可完成下载.

尽管下载流程简单，但 KDFS 作为存储服务最大的服务压力主要源自不同应用对相同文件或统一 K-NODE 文件的请求下载. 为了避免某一个 K-NODE 负荷太大，本文基于动态 WLC 算法[28]及静态 IP 哈希算法实现了多 K-NODE 负载均衡，以实现下载分流和保证大文件断点下载[29]的一致性. 总的来说，

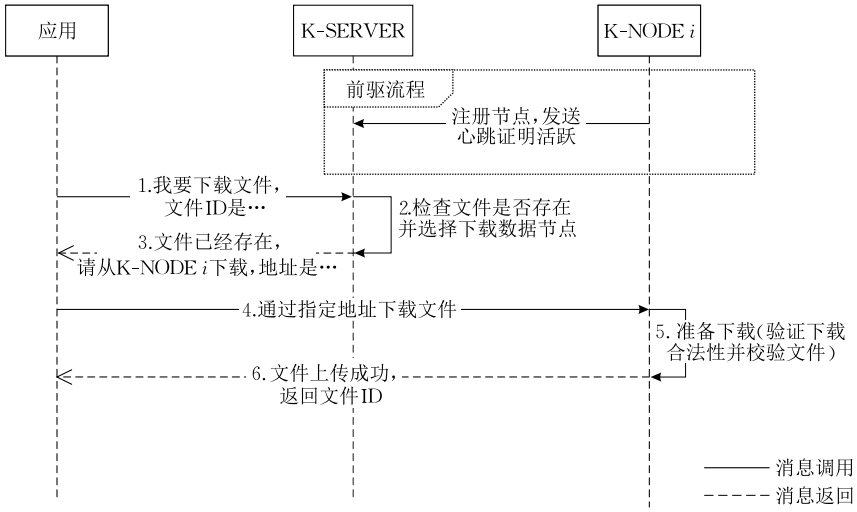


图 7　询问下载流程

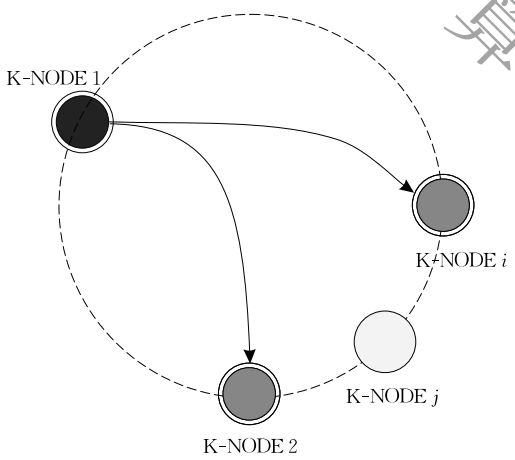
下载优先级遵循“加权就近”+“加权连接数”,当距离不敏感时或最近 K-NODE 负载过高需要牺牲就近特性时,用户会收到一个连接数最少的下载链接.实际上文件上传也遵循文件下载的负载均衡规则,但“加权连接数”的权值会更高,因为大文件上传需要长时间连接,会占用较大带宽,维护可用的带宽是提供优质上传体验的保证.

5 动态组网与文件最优分布

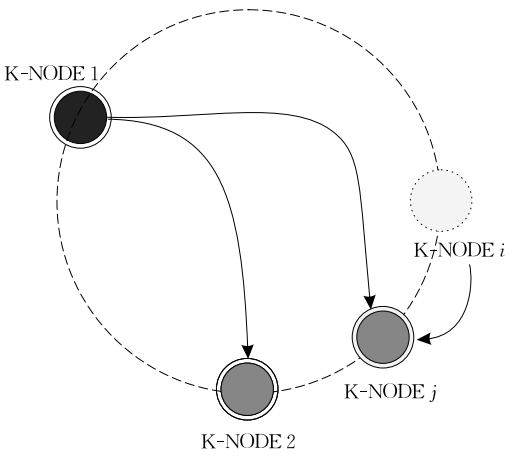
5.1 动态组网策略

K-NODE 网络动态组网的过程主要包括三个步骤:K-NODE 节点的并网与退网发现、网络拓扑及时调整以及新的网络拓扑下文件的迁移.

图 8 描述了一个文件在 K-NODE 网络状态稳定的情况下其副本的分布,以及当其中一个存储节点 K-NODE[i]退网后,文件从该节点向 K-NODE[j]节点迁移的过程.



(a) 某文件在某稳定状态下KDFS中的分布



(b) 节点i退网后KDFS动态组网并发生文件迁移

图 8 KDFS 文件分布及动态组网过程

5.1.1 存储节点的并网与退网发现

新的存储节点并网加入 K-NODE 网络或已在网运行的存储节点退出 K-NODE 网络是动态组网的充要条件.K-NODE 节点的并网与退网的发现均通过注册制实现.

在并网发现方面,任何一个 K-NODE 节点并入网络的时候都明确已知 K-SERVER 的网络位置,从而能够主动向其发送注册请求,请求信息内容包括 K-NODE 网络位置、当前物理存储状态、网络带宽容量及工作负载情况等.K-SERVER 接收状态并综合全网情况考虑决定是否允许其并网运行,一般情况下状态良好的节点都会被自动放行.

在退网发现方面,K-NODE 节点并网后被要求按固定间隔时间向 K-SERVER 发送心跳以证明其健康存在并运行良好,一旦 K-SERVER 在特定时间内接受不到该心跳信息或者认为该节点状态不足以满足当前在网需求时,会操作该节点退网.

5.1.2 网络拓扑实时调整

当并网或退网被发现后,K-NODE 网络将被要求立即进行调整.新的 K-NODE 节点并网调整过程如下:

(1) K-SERVER 将包括新的 K-NODE 在内所有节点位置全网广播到已在网运行的 K-NODE 节点;

(2) 每个 K-NODE 节点主动探知与其他 K-NODE 节点的最短网络距离(节点之间的通讯时间),并在间隔时间内动态维护;

(3) 此时新的 K-NODE 已经可以接受就近文件上载,为了满足就近文件下载需求,按照 K-SERVER 文件资源池的指导向邻近的节点同步文件,具体包括:

① K-SERVER 告知新的 K-NODE 节点需要同步文件清单,并标明文件所处的具体 K-NODE 节点;

② 新的 K-NODE 节点根据维护与其他 K-NODE 最短网络距离,从最近的 K-NODE 上去获取待同步文件;

③ 文件同步完毕报备 K-SERVER 文件资源池,完成后即可提供下载服务.

在网 K-NODE 节点退出网络调整过程步骤(1)和步骤(2)与并网流程一致,接下来:

(4) K-SERVER 检查文件资源池发现受存储节点退网影响的文件;

(5) K-SERVER 根据文件最优分布算法计算并确定受影响文件新的副本所需要存储的新的 K-NODE 节点;

(6)被指派接受受影响文件新副本的 K-NODE 节点根据按照 K-SERVER 文件资源池的指导向邻近的节点同步文件.

5.1.3　存储节点间文件同步

图 9 描述了文件从存储节点 K-NODE[i]向 K-NODE[j]同步的过程. 包括 K-SERVER 根据文件弹性存储策略计算文件副本应该同步到哪几个关键 K-NODE,并通知这些待同步文件副本的 K-NODE 准备同步文件,并且应该从哪个目标 K-NODE 上去获得文件;K-NODE 收到 K-SERVER 文件同步指示后将待同步的文件加入到同步队列;K-NODE 从同步队列中取出待同步的文件信息并向目标 K-NODE 请求获取文件,得到同意后开始同步文件;K-NODE 同步完成文件后向 K-SERVER 报告文件同步完成;K-SERVER 维护文件资源池,更新对应逻辑文件的最新物理存储地址.

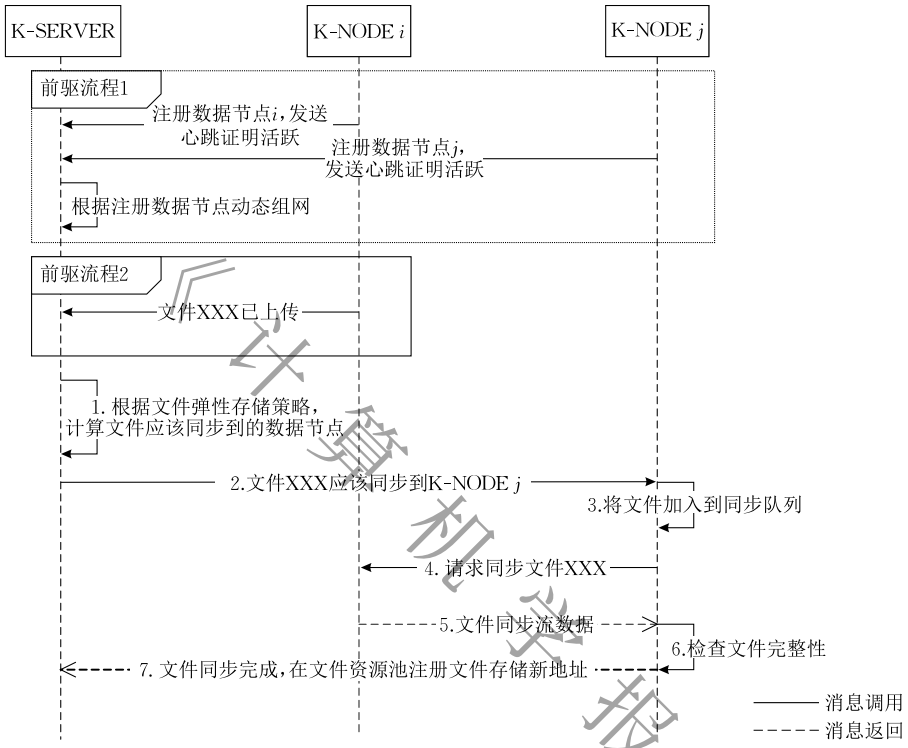


图 9　文件在 K-NODE 间同步流程

文件在 K-NODE 之间同步是 KDFS 内部机制,对用户透明. 根据 KDFS 架构及相关机制,一旦文件上传到 KDFS 且无论内部同步是否完成,应用即可直接从 KDFS 上下载文件,只是可能当用户从一个地区上传马上又需要从另外一个地区下载时,下载的体验会有所折扣. 因此,KDFS 对文件在 K-NODE 之间的同步时效性并没有硬性规定,在实际生产环境中,一般的策略是近实时同步或数据中心网络带宽闲置同步. 前者只要确定需要进行文件同步就开始按照同步队列优先级进行 K-NODE 之间文件传输,这种策略不考虑带宽利用的经济性;后者则一般选择在半夜或凌晨等网络带宽峰值较低的时候同步,对带宽的利用率达到最佳.

5.2　基于聚类的文件最优分布

文件最优分布的目标是在确保任意文件在 KDFS 内跨数据中心实现冗余备份的同时,备份文件存储在合适 K-NODE 节点向外提供服务,且服务覆盖的范围最广;当由于 K-NODE 节点并网或退网而发生网络重组时,备份文件能够从原本合适的源存储节点同步到新的网络环境下的目标存储节点. 文件最优分布的关键问题是解决在当前网络状态下,文件最优的存储位置问题. 已知网络重组前文件存储的 K-NODE[i]节点,并能够计算当网络发生变化后文件最适合存储的 K-NODE[j]节点,即可得出文件的同步路径,并通过同步机制快速实现动态组网后文件的再次最优分布.

如图 10 所示,在 KDFS 面向较大一块区域提供服务的时候,为了确保不同地区的用户体验往往会在相应地点设置 K-NODE 节点,K-NODE 节点具有明显的位置特性. 根据文件弹性存储策略,当一个源文件上传到相应的 K-NODE 节点,不可能在其他每个节点均存储一份文件备份,而必须在所有节

点中进行选择,并且要求选择的节点能够提供最大的服务范围.

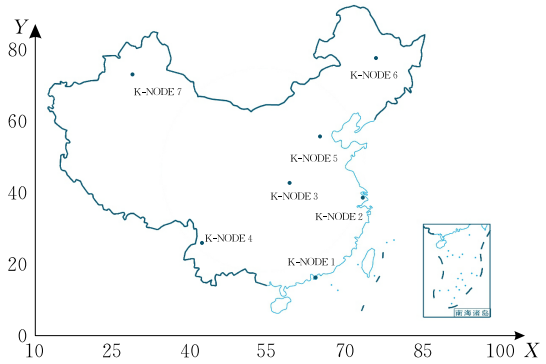


图 10 基于聚类的文件最优分布

在进行数学建模过程中,我们将所有的 K-NODE 节点认为是某坐标系中的一个个点,其中 X 轴代表纬线 Y 轴代表经线而每个坐标点的值则表示其在坐标系中的相对的位置. 这样当文件的弹性存储策略计划在拥有 m 个存储节点的 KDES 中存储 n 份时,文件的最优分布问题就变成了需要把坐标系中的 m 点分成 n 份,每份之间要有明显的位置差异性,这是一个典型的聚类问题. 在将 m 个存储节点分为 n 个群组之后,只需在每个群组中选择当前负载最小的节点存储该文件即可.

我们基于 ISODATA 算法来实现 K-NODE 网络中存储节点的聚类. ISODATA 是一种基于 k-means 的聚类算法,由于 k-means 算法通常适用于较多样本的情况,而在我们的情况下样本个数较少,因此必须对中心点的选取加一些限制,以避免极端情况出现. 和 k-means 中的随意选取初始中心点不同,一旦初始中心点选的不好,群组的划分极有可能很不合理,从而失去群组划分的意义,因此我们在选取初始中心点的时候,使他们之间的距离尽量大,从而达到分散选取中心点的目的. 此外,算法继承了 ISODATA 的中心节点合并和分裂的过程,使算法能够自适应地调整群组个数,达到合理分配的效果.

算法 1. 基于ISODATA 的K-NODE聚类算法.

输入:TE(类内部距离阈值上限)、TC(类中心间距阈值下限)、c(初始类个数)、$nodes$(节点的集合)、TM(精度阈值)

输出:$clouds$(二维数组,每一行包含该群组所有节点)

```
def K_Node_Cluster(TE,TC,NS,c,nodes,TM):
    MAX_VALUE=1000000
    cores=set([])
    index=random.randint(0,len(nodes))
    cores.add(index)
    WHILE len(cores)<c:
    //每次选择与所有已选定中心点距离之和最大的点作为新的中心点
        smallestInterval=MAX_VALUE
        nodeChosen=0
        FOR i in range(nodes):
            IF i in cores:
                CONTINUE
            interval=0
            FOR j in range(cores):
                interval=interval+
                    ((nodes[i].x-nodes[j].x)**2+
                    (nodes[i].y-nodes[j].y)**2))**0.5
            IF interval<smallesInterval:
                smallestInterval=interval
                nodeChosen=i
        cores.add(i)
    lastR=0
    mark=true
    clouds=[[] for i in range(c)]
    WHILE mark:
        clouds=[[] for i in range(c)]
        //将每个节点分配到距其最近的中心所在的群组
        r=0
        FOR i in range(len(nodes)):
            interval=MAX_VALUE
            class=0
            FOR j in range(len(cores)):
                t=((nodes[i].x-nodes[j.id].x)**2+
                    (nodes[i].y-nodes[j.id].y)**2))**0.5
            IF t<interval:
                interval=t
                class=j
        clouds[class].append(nodes[i].id)
            r=r+interval**2
        IF abs(r-lastR)>TM:
            //根据精度判断是否收敛
            mark=true

        //重新寻找中心点
        FOR i in range(len(cores)):
            x=0
            y=0
            FOR j in range(clouds[i]):
                x=x+nodes[j].x
                y=y+nodes[j].y
                mark=true
            x=x/len(clouds[i])
```

```
        y=y/len(clouds[i])
        cores[i].x=x
        cores[i].y=y
    //判断是否符合合并的条件
    merge=false
    FOR i in range(len(cores)):
      IF merge:
        BREAK
      FOR j in range(len(cores)):
        IF i==j:
          CONTINUE
        interval=((cores[i].x-cores[j].x)**2+
                  (cores[i].y-cores[j].y)**2)**0.5
    IF interval<TC:
      cores[i].x=(cores[i].x+cores[j].x)/2
      cores[i].y=(cores[i].y+cores[j].y)/2
          del cores[j]
          merge=true
          mark=true
          BREAK
    //判断是否符合分裂的条件
    split=false
    FOR i in range(len(cores)):
      IF split:
        BREAK
      FOR j in range(clouds[i]):
        interval=((nodes[j].x-cores[i].x)**2+
                  (nodes[j].y-cores[i].y)**2)**0.5
        IF interval>TE:
          cores.append(nodes[j])
          mark=true
          split=true
          BREAK
return clouds
```

上述算法一方面继承了 k-means 的迭代直至收敛的做法，另一方面通过减少初始点的随机性保证分组的稳定性. 同时合并和分裂的机制也摆脱了必须先判断出分组个数的桎梏，基本满足了问题的需求. 然而，这个算法的效果也同样依赖参数的选取. 以下是一些关于该算法参数设定的建议.

参数选取中比较重要的有初始中心节点个数 c，建议选取$[(n-1)/3]$，若太多了会导致中心节点初始选取均匀效果不佳，同时每个群组点数过小，达不到群组的目的，若太少则过分依赖分裂过程，对于 TE 选值过分依赖，容错性较差.

TM 的选择决定了算法的收敛，通常选择 1 即可，精度太高不易收敛，由于问题的特殊性我们不能通过迭代次数强行结束，因为不收敛的聚类对我们组网之后的同步等过程而言会带来过大的代价. 但同时对于分布比较密集的 K-NODE 具体分到哪个群组差别并不会特别大，因此选择 1 或者 10 这样偏大的参数即可.

对于 TE 和 TC 的选取依赖于节点的分散程度，若选择不当很可能导致群组过多，备份过量，节点负担过大或者群组过少，备份太少，导致 QoS 和安全性能的下降. 对于分散程度不同的情况，使用同一套 TE，TC 参数必然会导致结果不佳，因此我们认为 TE 和 TC 可以通过对节点数据的初步处理得到. 在不知道节点分布的情况下，我们需要考虑各种极端情况，但无论如何，至少应该有两个以上的群组，这样才有采用此策略的必要，因此 TE 的值设为各节点间距离的最大值的三分之一，这样可以保证至少两个群组的存在. TC 的设置也基于同样的思想，我们应当尽量避免一个节点为群组的情况出现，因此我们设 TC 为各节点间距离最小值的三倍，这样可以保证群组内部节点个数不至于太小. 但是这样的设置显然无法应对某些极端情况，比如所有的节点间均等距，这样的话，一方面要合并，另一方面又需要分裂，将陷入死循环. 因此，参数输入前必须检验，若 $TE<TC$，则可设置 TE，TC 均为$(TE+TC)/2$，如此可以保证不陷入死循环.

6　实现和实验

6.1　系统实现

基于上述架构、方法和策略，我们利用 Golang 语言及其 beego 框架开发实现了 KDFS，并利用 Docker 进行部署应用于生产. Golang 语言是谷歌倡导的新型开发语言，其强大的标准库、全开发环境和跨平台构建能力等比较适合于 KDFS 等高并发和强时序的后台系统开发. Docker 则是基于 Golang 语言开发的一款轻量级容器技术，它实现内核级别的资源共享，能够最大化地提升内存使用效率，最重要的是能够通过标准化的方式简单快速地实现集群的弹性部署.

基于文件冗余安全、面向全国提供最优服务线路以及成本的考虑，生产环境的 KDFS 主要部署了 2 个 K-SERVER 节点和 3 个 K-NODE 节点. 2 个 K-SERVER 组成 K-SERVER-HA 主要部署在广州 BGP 数据中心提供四线接入能力. 3 个 K-NODE 则分别部署在北京、深圳和广州数据中心通过 VPN 共同组网面向应用提供存储服务. 根据就近服务策

略,北京金山 K-NODE 主要用于服务华北及东北用户,广州 K-NODE 主要服务华南和西南用户,而深圳数据中心主要服务华南和华东用户.根据文件弹性存储策略,目前一份文件均保存 4 个备份分别位于 3 个 K-NODE 节点之中.

目前,KDFS 已经在金蝶旗下部分 ERP 产品和互联网产品,典型的产品包括"记账王(云盘版)"、"智慧记(云盘版)"以及"金蝶云盘"产品.

如图 11 监控系统截图展示了当前 KDFS 健康状态以相关运营数据.从健康状态看 KDFS 自上一次更新后已经正常运行 6 天 17 个小时,所有节点均运转良好.从运营数据看,当前 KDFS 已为应用存储了 274 万份文件,总计大小 4.4 T,基本上平均分布到了每一个节点,少量文件正处于节点之间的同步过程中.

图 11 生产环境 KDFS 部署及运营情况

综上,KDFS 可扩展服务能力及其跨数据中心集群部署得到初步体现.

6.2 存储性能实验

6.2.1 实验方案

为了验证跨数据中心集群部署的 KDFS 整体性能能够比同一数据中心内单集群部署的 HDFS 在面向应用服务方面提供更加优质的体验而设计本实验.

实验的方案是在深圳和北京两个数据中心分别架设对标用 HDFS 集群,各自称之为深圳对标 HDFS 和北京对标 HDFS,然后分别从深圳和北京两地对 KDFS(含深圳节点和北京金山节点)、北京对标 HDFS 和深圳对标 HDFS 进行同样大小和数量的文件上传实验.两个对标集群部署的环境同 KDFS 在深圳和北京金山数据中心环境和采样方法一致:

(1) Hadoop 2.7.0(同 K-NODE 封装的 HDFS);

(2) 服务器 10 台,均为 4 核 CPU,16 GB 内存,4 TB 硬盘;

(3) 同时段 3 次采样均值;

(4) 3 种不同负载的采样均值;

(5) 公共网络,非专线网络;

(6) 客户端设置(如文件校验、大文件分块上传及续传)相同.

实验 1. 从深圳分别向 KDFS、北京对标 HDFS 和深圳对标 HDFS 分别上传等量的 20 MB、40 MB、60 MB、80 MB 和 100 MB 文件(KDFS 对小文件的有优先处理机制,对超过 64 M 大文件有分块传输与存储、完整性校验等机制,为了让实验更具代表性因此选取上述大小文件作为样本),统计文件上传到这些集群的时间.

实验 2. 从北京分别向 KDFS、北京对标 HDFS 和深圳对标 HDFS 分别上传等量的 20 MB、40 MB、60 MB、80 MB 和 100 MB 文件,统计文件上传到这些集群的时间.

6.2.2 实验结果

实验 1 结果如图 12 所示,相同数量不同大小的文件从深圳分别上传 KDFS、北京对标 HDFS 和深圳对标 HDFS 的上传采样均值结果显示:

(1) 相同文件上传到 KDFS 深圳 K-NODE 的时间与上传到深圳对标 HDFS 的时间大体一致;

(2) 相同文件上传到 KDFS 深圳 K-NODE 的时间小于上传到北京对标 HDFS 的时间;

(3) 相同文件上传到北京对标 HDFS 的时间要小于文件成功上传到 KDFS 深圳 K-NODE 然后同步到北京 K-NODE 所花费的时间.

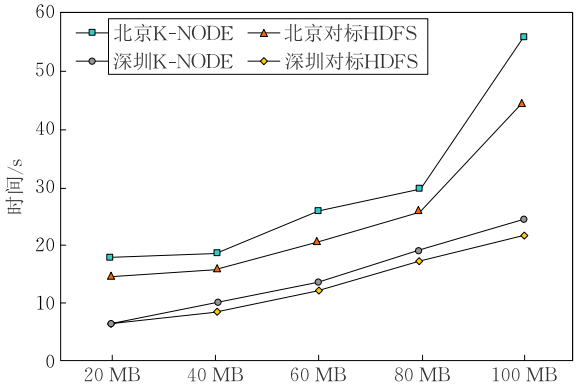


图 12 实验 1 文件从深圳上传的结果

实验 2 结果如图 13 所示,相同数量不同大小的文件从北京分别上传 KDFS、北京对标 HDFS 和深圳对标 HDFS 的上传采样均值结果显示:

(1) 相同文件上传到 KDFS 北京 K-NODE 的时间与上传到北京对标 HDFS 的时间大体一致;

(2) 相同文件上传到 KDFS 北京 K-NODE 的时间小于上传到深圳对标 HDFS 的时间；

(3) 相同文件上传到深圳对标 HDFS 的时间要小于文件成功上传到 KDFS 北京 K-NODE 然后同步到深圳 K-NODE 所花费的时间.

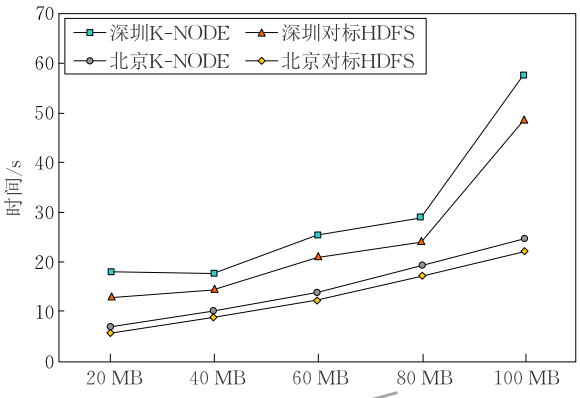


图 13　实验 2 文件从北京上传的结果

对两个实验数据进行比较，相同数量不同大小的文件分别从北京和深圳上传 KDFS 采用均值结果如图 14 所示.

(1) 相同文件从北京上传到北京 K-NODE 和经由北京 K-NODE 同步到深圳 K-NODE 相差较大；

(2) 相同文件从深圳上传到深圳 K-NODE 和经由深圳 K-NODE 同步到北京 K-NODE 相差较大；

(3) 相同文件从北京上传到北京 K-NODE 节点时间与从深圳上传到深圳 K-NODE 节点时间相差不大；

(4) 相同文件从北京经由北京 K-NODE 节点同步到深圳 K-NODE 节点时间与文件从深圳经由深圳 K-NODE 节点同步到北京 K-NODE 节点时间相差不大.

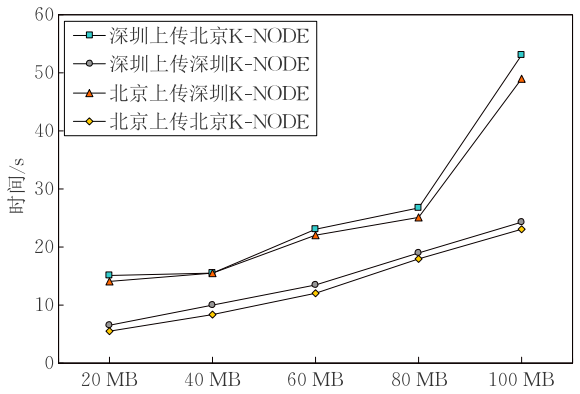


图 14　相同文件分别在北京、深圳上传 KDFS 结果

6.2.3　结果分析

从实验 1 和实验 2 的结果看，无论是从深圳还是北京上传不同大小文件到 KDFS，结果上传耗时是相近的，这是因为 KDFS 在深圳和北京都部署有 K-NODE，每次文件的上传其实都是上传到本地的存储节点，然后再通过 KDFS 内部的同步机制将文件同步到其他节点，这个跟将相同的文件上传到本地对标 HDFS 系统中的耗时是类似的，因为每一个 K-NODE 节点实际上就是对 HDFS 系统的封装，因此在网络环境相似情况下，性能不会有太大优劣. 而从深圳或北京上传到异地对标 HDFS 的耗时可以明显看出，这个耗时是明显大于同样的文件上传到 KDFS 的.

但由于 HDFS 难以实现跨数据中心部署，而 KDFS 能够在全国多个地区的数据中心部署存储节点，因此可以得出结论：相同网络环境下，同样的文件上传到 KDFS 的速度体验要整体优于部署到 HDFS 中；下载虽然没有进行比较实验，但其原理跟上传类似，因此可以推理出相同网络环境下，同样文件从 KDFS 下载要整体优于 HDFS.

需要注意的是，由于生产环境 KDFS 针对超过 64 MB 以上大文件上传与 KDFS 内部同步有文件完整性校验、分块存储以及局部加密，以及小文件优先存储等措施，导致在所有实验中，80 MB 至 100 MB 区间，传输时间会有明显上升，但对实验结果对比分析和实验结果并不造成影响.

6.3　动态组网实验

6.3.1　实验方案

为了验证当 KDFS 各个节点在协同服务的时，有新的 K-NODE 节点加入或有已并网使用的 K-NODE 节点因故退出时动态组网算法的有效性和效率，设计本实验.

实验直接在 KDFS 生产环境中进行，环境同实验 1 和实验 2，通过在北京、广州和深圳三个现有的 K-NODE 节点基础之上，额外增加江苏宿迁一个 K-NODE 节点，并在江苏南京设置客户端应用进行文件上载实验.

实验 3. 在相同环境下 3 小时内从南京持续上载 20 MB 和 80 MB(KDFS 对 64 MB 以上文件会进行分块传输和存储)文件到 KDFS，在第 1 小时仅设置北京、广州和深圳 3 个 K-NODE 节点，从第 2 个小时开始的时候将宿迁 K-NODE 节点并入 K-NODE 网络，从第 3 个小时开始将宿迁以及深圳 K-NODE 节点从 K-NODE 网络人工移除.

6.3.2　实验结果

实验 3 结果如图 15 所示. 在第 1 个小时内分别上载 20 MB 和 80 MB 文件耗时均在 28 s 和 110 s

左右波动；当第 2 个小时有新的存储节点并入 K-NODE 网络后，两类不同大小文件的耗时在一段时间内明显下降，最后分别稳定在 20 s 和 80 s 上下；当第 3 个小时有在网的存储节点退出 K-NODE 网络后，两类不同大小文件耗时陡增，恢复到第 1 个小时耗时状态.

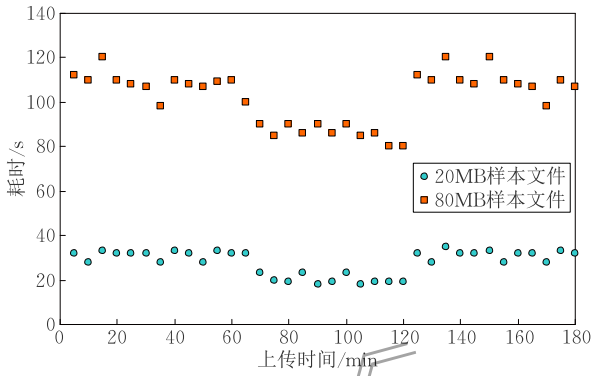


图 15 KDFS 动态组网(并网、退网)时耗时比较

6.3.3 结果分析

结果显示，以一个小时为时间段，每个时间段内不同大小文件上传都相对稳定，在第 2 个小时开始当位于宿迁的 K-NODE 节点并网后，网络发生动态调整，位于南京的文件上载被引导存入宿迁节点，在网络调整期间，文件的上传消耗时间平滑下降，网络调整完毕后，上载耗时趋于相对稳定. 而当宿迁数据节点和深圳节点移出 K-NODE 网络后，从南京上载的文件突然失去就近上载数据节点，此时从南京上传的文件被迅速引导到其他相对较近点，因此当节点退网后，耗时很快陡增，并逐渐趋于稳定，总体跟第 1 个小时耗时持平.

结果还表明由于超过 64 MB 的文件涉及到分块上传和存储，80 MB 文件的上传稳定性要弱于 20 MB 文件上传的稳定性，值得 KDFS 进一步改进.

7 结 论

本文针对 HDFS 集群主要局限于同一数据中心内部部署且可扩展性受限的问题，通过架构设计和关键技术研发，提出一种跨数据中心集群部署的文件资源调度机制并实现了金蝶分布式文件服务 KDFS. 较之于传统的 HDFS 集群，KDFS 能够支持多个 HDFS 集群跨数据中心部署、动态组网并协同面向多应用提供透明服务. 实验和实践证明，跨 HDFS 集群的文件资源调度机制不但解决了 HDFS 集群可扩展性受限的问题，同时通过跨数据中心部署，实现了集群文件异地冗余灾备、全局范围最优分布从而为文件实现就近存取服务，有效地提高了用户使用 KDFS 的体验.

但 KDFS 依然有较大可改进空间，主要的研究点包括 KDFS 去中心化，包括去中心化后文件资源池的分布式管理、节点的并网与退网发现以及基于对等网络的文件最优分布算法优化等，从而进一步提升 K-NODE 网络服务能力等.

致 谢 本文作者感谢金蝶软件(中国)有限公司大数据云平台部产品、开发、测试和运维团队在 KDFS 研发和测试方面的努力和辛勤工作！

参 考 文 献

[1] Howard J H, Kazar M L, Menees S G, et al. Scale and performance in a distributed file system. ACM Transactions on Computer Systems, 1988, 6(1): 51-81

[2] Weil S A, Brandt S A, Miller E L, et al. Ceph: A scalable, high-performance distributed file system//Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation. Seattle, USA, 2006: 307-320

[3] Ghemawat S, Gobioff H, Leung S T. The Google file system. ACM SIGOPS Operating Systems Review, 2003, 37(5): 29-43

[4] Shvachko K, Kuang H, Radia S, et al. The Hadoop distributed file system//Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies. Incline Village, USA, 2010: 1-10

[5] Borthakur D, Gray J, Sarma J S, et al. Apache Hadoop goes realtime at Facebook//Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data. Athens, Greece, 2011: 1071-1080

[6] Wang L, Tao J, Ranjan R, et al. G-Hadoop: MapReduce across distributed data centers for data-intensive computing. Future Generation Computer Systems, 2013, 29(3): 739-750

[7] Luo Y, Luo S, Guan J, et al. A RAMCloud storage system based on HDFS: Architecture, implementation and evaluation. Journal of Systems and Software, 2013, 86(3): 744-750

[8] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data//Proceedings of the 7th USENIX Symposium on Open System Design and Implementation. Seattle, USA, 2006: 205-218

[9] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Communications of the ACM, 2008, 51(1): 107-113

[10] Dong B, Qiu J, Zheng Q, et al. A novel approach to improving the efficiency of storing and accessing small files on Hadoop: A case study by PowerPoint files//Proceedings of the 2010 IEEE Services Computing Conference. Miami, USA, 2010: 65-72

[11] Liu X, Han J, Zhong Y, et al. Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS//Proceedings of the 2009 IEEE International Conference on Cluster Computing. New Orleans, USA, 2009: 1-8

[12] Abad C L, Lu Y, Campbell R H. DARE: Adaptive data replication for efficient cluster scheduling//Proceedings of the 2011 IEEE International Conference on Cluster Computing. Austin, USA, 2011: 159-168

[13] Hong Xu-Sheng, Lin Shi-Ping, Efficiency of storaging small files in HDFS based on MapFile. Computer Systems & Applications, 2012, 21(11): 179-182(in Chinese)
(洪旭升，林世平. 基于 MapFile 的 HDFS 小文件存储效率问题. 计算机系统应用，2012，21(11)：179-182)

[14] Chandrasekar S, Dakshinamurthy R, Seshakumar P G, et al. A novel indexing scheme for efficient handling of small files in Hadoop distributed file system//Proceedings of the 2013 IEEE International Conference on Computer Communication and Informatics. Tamilnadu, India, 2013: 1-8

[15] Luo Li. A road for Yunti to crossing data centers. Programmer, 2013, 12: 110-113(in Chinese)
(罗李. 云梯的多 Namenode 和跨机房之路. 程序员，2013，12：110-113)

[16] Shen Hong, YARN cluster on Alibaba Yunti. Programmer, 2013, 11: 105-107(in Chinese)
(沈洪. 深入剖析阿里巴巴云梯 YARN 集群. 程序员，2013，11：105-107)

[17] Zhang L J, Cai H, Zhang J. Services Computing. Beijing: Tsinghua University Press, 2007

[18] Jiang L, Li B, Song M. THE optimization of HDFS based on small files//Proceedings of the 2010 IEEE International Conference on Broadband Network and Multimedia Technology. Beijing, China, 2010: 912-915

[19] Kiran M, Kumar A, Mukherjee S, et al. Verification and validation of MapReduce program model for parallel support vector machine algorithm on Hadoop cluster. International Journal of Computer Science Issues, 2013, 10(1): 317-325

[20] Di Martino B, Aversa R, Cretella G, et al. Big data (lost) in the cloud. International Journal of Big Data Intelligence, 2014, 1(1-2): 3-17

[21] Dekel E, Sahni S. Parallel generation of postfix and tree forms. ACM Transactions on Programming Languages and Systems, 1983, 5(3): 300-317

[22] Zawodny J D, Balling D J. High Performance MySQL: Optimization, Backups, Replication, Load Balancing & More. O'Reilly Media, Inc., 2004

[23] Qian J, Liao L. Realization of dynamic floating IP cluster based on keepalived. Control and Instruments in Chemical Industry, 2012, 7: 0-30

[24] Godfrey P, Stoica I. Heterogeneity and load balance in distributed hash tables//Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Miami, USA, 2005: 596-606

[25] Kiselyov O. A network file system over HTTP: Remote access and modification of files and files//Proceedings of the USENIX Annual Technical Conference. Monterey, USA, 1999: 75-80

[26] Hu B, Zhang L J, Liu D, et al. A cloud oriented account service mechanism for SME SaaS ecosystem//Proceedings of the 2012 IEEE International Conference on Services Computing. Honolulu, USA, 2012: 336-343

[27] Hunt P, Konar M, Junqueira F P, et al. ZooKeeper: Wait-free coordination for Internet-scale systems//Proceedings of the USENIX Annual Technical Conference. Boston, USA, 2010: 9-17

[28] Zhang W. Linux virtual server for scalable network services //Proceedings of the Linux Symposium. Ottawa, Canada, 2000: 10-17

[29] Chen H, Zhang L J, Hu B, et al. On developing and deploying large-file upload services of personal cloud storage//Proceedings of the 2015 IEEE International Conference on Services Computing Conference. New York, USA, 2015: 371-378

HU Bo, born in 1983, Ph. D., researcher. His research interests include cloud computing and big data.

CHEN Huan, born in 1984, Ph. D. His research interests include cloud computing and big data.

ZHANG Liang-Jie, born in 1969, Ph. D. His research interest is service computing.

MOU Jian-Wei, born in 1988, bachelor, engineer. His research interest is cloud storage.

DAI Guang-Li, born in 1995, bachelor, engineer. His research interest is machine learning.

MA Yu-Tao, born in 1980, Ph. D., associate professor, His research interest is service computing.

Background

In big data era, Hadoop Distributed File System (HDFS) is a most widely used clustered file system, which is designed to be built onto commodity hardware. Compared with the other storage systems, it is characterized by its high fault tolerance, which makes it suitable for building onto low-cost storages or even legacy storages. Meanwhile, it provides capability of high throughput in accessing huge volume of data, which makes it powerful at handling those applications with large data sets.

HDFS provides reusable key storage modules for commercial storage, largely reducing development cost. However, it still has two problems: the scalability of clustering the support of multi-applications and transparent services. Without optimization, HDFS stores small files in an uneconomical way. Many recent works tried to address the small files storage problem but they did not solve the root cause of the problem. In fact, it can be viewed as a scalability problem of HDFS. Some researchers tried to use HDFS Federation to slicing several Namespaces across data centers. In contrast, this paper proposes solutions of coordinating several HDFS with multiple Namespaces to completely solve the scalability problem. In addition, this paper proposes solutions for the subsequent problems after building inter-data center HDFS cluster. For example, the problem of file distribution optimization, and file migration across multiple HDFSs.

The outcome of this research is the Kingdee Distributed File System (KDFS) and the cloud storage application, which we named KDrive. These two services have already served millions of users. This research work has been studied for three years and it is developed by two teams of Kingdee: Department of Kingdee Bigdata & Cloud and Kingdee Research Institute.

This work is supported by the National Key Research and Development Plan (Project No. 2016YFB1000800), the Central Grant Funded Cloud Computing Demonstration Project of China, R&D and Industrialization of the SME Management Cloud (Project No. [2011]2448), the Fund for Leading Talents of Guangdong Province (Project No. [2012]342), the Shenzhen High-Tech Project, under Grant No. FWYCX20140310010238.