计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(11):64−70 [doi: 10.15888/j.cnki.csa.006663]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Hadoop 的遥感影像业务管理系统设计①

张　扬, 谢　彬, 王敬平, 唐　鹏

(华东计算技术研究所, 上海 201808)
通讯作者: 张　扬, E-mail: cruise0716@163.com

摘　要: 随着信息技术、航天技术的不断发展,遥感影像数据量呈现爆发性增长. 针对陕西某测绘所管理影像数据时出现的存储需求量大、读取过程缓慢、图像处理耗时长等问题. 本文基于 LFU (Least Frequently Used) 缓存算法, 设计了三级缓存模块, 用来高速缓存影像数据. 同时结合 Hadoop 开源技术, 设计了遥感影像业务管理系统. 该系统具有高内聚, 低耦合, 扩展性强的特点. 经过系统性能测试, 该系统能有效解决影像存储瓶颈, 同时能提升影像读取速度, 缩短影像处理时间.

关键词: 遥感影像; 存储瓶颈; LFU 算法; 三级缓存模块; Hadoop

引用格式: 张扬,谢彬,王敬平,唐鹏.基于 Hadoop 的遥感影像业务管理系统设计.计算机系统应用,2018,27(11):64–70. http://www.c-s-a.org.cn/1003-3254/6663.html

Design of Remote Sensing Image Management System Based on Hadoop

ZHANG Yang, XIE Bin, WANG Jing-Ping, TANG Peng

(East China Institute of Computing Technology, Shanghai 201808, China)

Abstract: With the continuous development of information technology and aerospace technology, the amount of remote sensing image data has shown an explosive growth. The problem of large storage requirements, slow reading process, and long image processing time has been faced with the management of image data by a surveying and mapping facility in Shaanxi. Cache algorithm is designed with a three-level cache module used for cache image data. At the same time, combined with Hadoop open source technology, a remote sensing image business management system is designed. The system has high cohesion, low coupling, and strong scalability features. Testing shows that the system can effectively solve the image storage bottleneck, while improving the image reading speed and shorten the image processing time.

Key words: remote sensing image; storage bottleneck; LFU algorithm; three-level cache module; Hadoop

1　问题分析

遥感影像作为一种特殊的数字图像, 具有高分辨率, 高覆盖范围, 信息量大等特点. 在环境监测, 资源探测, 灾害防护, 城市规划, 军事行动等方面具有极高的应用价值[1].

陕西某测绘所近些年一直使用单机文件系统如 NTFS, 进行遥感影像数据存储. 随着影像数据不断积累, 数据规模达到海量级别, 传统解决方案不能满足海量存储需求, 同时扩展性很差, 不支持多用户数据共享. 与此同时, 科研人员在进行业务处理时, 发现影像读取速度缓慢, 进行影像聚类处理时, 等待周期较长.

针对以上问题, 本文基于 Hadoop 和三级缓存技术, 设计了一套遥感影像业务管理系统, 提供影像存储, 算法注册, 分布式计算的功能. 系统测试结果表明, 影像读取速度提升 2 倍以上, 计算处理时间缩短三分之一以上.

① 收稿时间: 2018-04-25; 修改时间: 2018-05-17; 采用时间: 2018-05-24; csa 在线出版时间: 2018-10-24

© 中国科学院软件研究所 http://www.c-s-a.org.cn

本文将介绍以下内容: 第 2 节介绍相关研究状况, 第 3 节是遥感影像管理系统架构设计, 第 4 节介绍四个核心模块的实现原理, 第 5 节给出系统测试方法和结果.

2 相关研究状况

关于如何存储和管理遥感影像数据, 国内外学者和商业技术公司都进行了很多相关的研究. 这些研究内容主要分为以下三种方式:

1) 采用传统文件系统, 如 NTFS, EXT, FAT, HFS 等. 首先将影像数据瓦片化, 之后将瓦片数据根据空间位置信息, 按照特定的目录结构进行存储. 这种方式成本低廉, 使用简单, 但是容错性和扩展性差, 不支持多用户数据共享, 并发操作等.

2) 采取关系型数据库 (RDBMS), 如 MySQL, Oracle 等. 首先将影像数据进行分块, 之后将数据块转化成二进制大对象 (Binary Large Object), 进行入库管理. 这种方式查询方式简单, 支持多用户数据共享. 但是关系型数据库对大对象 (Blob) 存取效率不高. 其次, 单个 Blob 对象一般最大支持 4 GB 的大小, 限制了影像数据块规模.

3) 采用集中式文件系统, 如 ArcGIS Image Server 等. 这种方式技术成熟, 易于维护. 但是这种方式的扩展性较差, 同时随着用户规模的增加, 中央服务器很容易达到性能瓶颈.

本文采用 HDFS 分布式存储策略, 将系统构建在廉价的 PC 机之上, 具有高度的容错性, 可扩展性, 易于维护的特点[2,3]. 同时基于 LFU 缓存算法, 设计了三级缓存模块, 用来高速缓存影像数据. 两种方式相互结合, 前者解决海量存储的需求, 后者实现影像数据的快速读取.

除此之外, 关于如何处理影像数据, 传统的解决方案是将处理算法运行在单一 PC 机之上. 但是很多影像处理算法如 K 均值、ISODATA 等, 可以结合 Map-Reduce 这种分布式处理框架进行改进[4–7]. 本文设计了算法注册和处理模块, 支持单机处理和 MapReduce 分布式处理两种运行模式.

3 系统架构

如图 1 所示, 系统采用 B/S 架构设计, 为用户提供友好的 Web 客户端. 方便用户进行影像导入, 计算处理等操作.

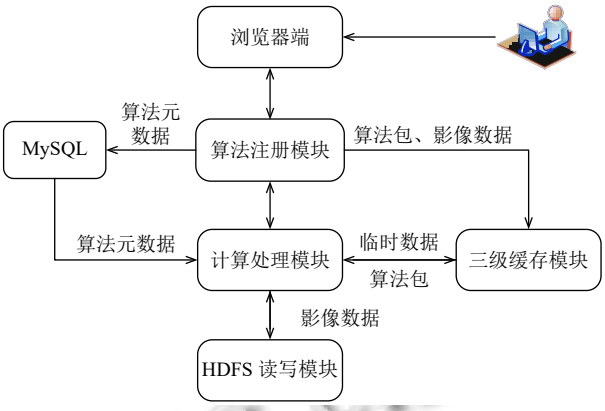

图 1 系统架构设计

整套系统分为四个核心模块, 分别是三级缓存模块, HDFS 读写模块, 算法注册模块以及计算处理模块. 满足软件设计高内聚, 低耦合的标准, 每个模块功能如下:

1) 三级缓存模块: 缓存影像数据块和计算处理模块产生的临时数据, 加速读取速度.

2) HDFS 读写模块: 构建 HDFS 集群, 对外提供读写接口. 满足遥感影像海量存储和多用户共享数据的需求.

3) 算法注册模块: 获取算法元数据信息, 导入 MySQL 数据库持久化.

4) 计算处理模块: 根据算法元数据, 开启单机处理模式或者 MapReduce 分布式模式.

影像数据写入时, 首先浏览器客户端与三级缓存模块和 HDFS 读写模块交互, 确定写入路径. 之后将影像数据从本地同时写入三级缓存模块和 HDFS 集群. 最后客户端端接收到成功写入信号, 关闭写入工作流. 否则, 重新发起写工作.

影像数据读取时, 浏览器客户端首先与三级缓存模块交互, 查找影像数据. 如果查找成功结束读取工作, 否则继续与 HDFS 底层存储模块交互, 查找数据块.

算法注册时, 浏览器客户端将算法包元数据写入 MySQL, 算法包同时写入三级缓存模块和 HDFS 集群. 算法运行时, 根据算法元数据, 开启对应算法运行模式.

4 模块原理和实现

4.1 三级缓存模块

三级缓存模块在系统中功能为以下两点: (1) 缓存影像数据, 提高影像读取速率. (2) 缓存影像处理过程中产生的临时数据, 减少影像处理时间.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

根据上述设计目标, 本模块架构设计采用主从设计思想, 共分为四个部分: Master、Salve、Client 以及切割预处理. 如图 2 所示.

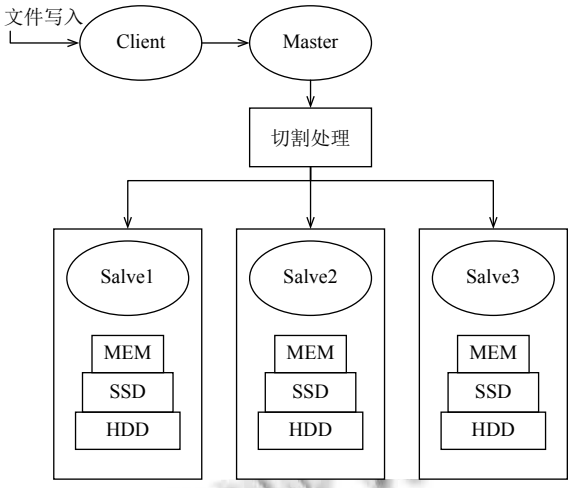

图 2 三级缓存模块架构

当影像文件写入 Salve 节点之前, 先对文件进行切割预处理, 将影像切割成很多 128 MB 大小相同的数据块, 不足 128 MB 的数据按实际大小成块. 例如一个 500 MB 的影像文件, 会切割成三个 128 MB 和一个 116 MB 大小的数据块.

Master 负责影像文件元数据管理. 元数据采用 Inode Tree 形式, 每个影像文件都是一个 Inode. 每个 Inode 记录着: 文件对应的数据块 id、名称、大小, 存储节点位置等信息. 同时维护一张 Mount 表, 记录每个文件三级缓存系统路径与 HDFS 文件路径的映射关系. 例如三级缓存系统根路径"/"对应"hdfs://master:9000/ThreeCache/". 方便影像文件在三级缓存系统和 HDFS 集群中的映射管理.

Salve 负责存储影像数据块, 管理本地 MEM(内存)、SSD(固态硬盘) 和 HDD(硬盘), 三种硬件设备构成三层缓存结构, 其读写速度依次递减. 每个数据块根据 LFU (Least Frequently Used) 算法缓存机制, 计算出热度值. 数据块根据热度值, 依次递减排列, 缓存到 MEM、SSD、HDD 中.

如图 3 所示, 在 LFU 算法中[8], 每个数据块历史访问次数越多, 热度值就越高. 核心思想是根据历史访问频率来判定未来访问频率. 同一影像文件对应的所有数据块具有相同的热度值.

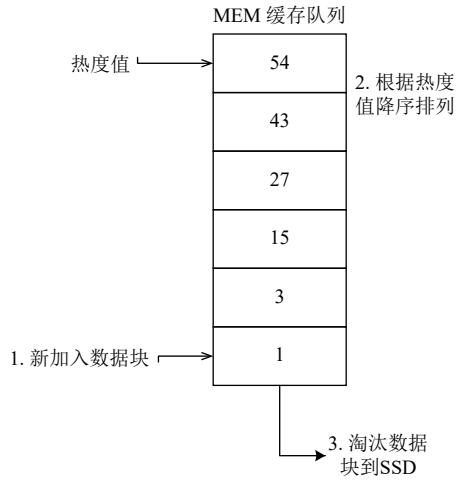

图 3 LFU 缓存机制算法

算法 1. LFU 算法

1. 新加入数据块, 放到缓存队列末尾. 初始热度值为 1.
2. 缓存队列中数据块每被访问一次, 热度值增加 1.
3. 队列重新排序, 热度值依次递减. 热度值相同的数据块, 最近访问时间靠后的, 排在前面.
4. 缓存队列满时, 加入新的数据块, 淘汰队列末尾数据块到低层次存储结构.
5. 低层次存储结构重复步骤 1、2、3、4.
6. 三级存储结构满时, 淘汰数据块.

Client 作为客户端, 负责向外提供文件写入和读取的访问接口. 当发生影像文件读取时, 三级缓存模块工作流程如下:

1) Client 接收到影像文件读取请求, 发送信息给 Master, 返回对应数据块位置信息.

2) 根据块位置信息, 在 Salve 中查找. 如果命中数据块, 就合并数据块成文件发送给对话发起模块, 之后 Client 结束对话.

3) 如果没有命中数据块, 则说明数据块被淘汰, 返回失败信息给 Client.

4) Client 当接收到返回的失败信息后, 返回查找失败信息给对话发起模块, 结束对话.

如果三级缓存系统中没有影像文件, 只能向 HDFS 发起 RPC 请求, 读取数据. 同时该影像文件重新加载到三级缓存系统, 更新热度值. 重新排列缓存系统中数据块位置.

4.2 HDFS 读写模块

HDFS 读写模块与三级缓存模块都具有影像数据存储功能. 但该模块功能在系统中作用是满足影像数

© 中国科学院软件研究所 http://www.c-s-a.org.cn

据的海量存储需求.

实现方案是构造 UploadFile, DownloadFile 方法, 方法中调用 Hadoop 提供 fileSystem 类 getFileStatus, copyFromLocalFile, open, delete 相关方法. 如图 4 所示, 当影像文件写入时, 该模块工作流程如下:

1) Web 前端获取影像名称, 初始路径, 目标路径. 发送写入请求给 HDFS 读写模块.

2) HDFS 读写模块接收写入请求后, 调用 UploadFile 方法获取初始路径, 和目标路径, 写入 configuration. 再调用 HDFS 的 copyFromLocalFile 接口.

3) HDFS 集群调用 DistributedFileSystem 对象, 的 create 方法, 返回 FSDataOutputStream 对象, 创建一个文件输出流.

4) 调用 DistributedFileSystem 对象, 与 NameNode 进行 RPC 调用, 在分布式文件系统创建目标路径.

5) 调用 FSDataOutputStream 对象, 向 DataNode 开始发起写入请求. 原始影像被分割成大小为 64k 的 packet, 包含校验码等发送出去.

6) DataNode 与其他 DataNode(副本放置) 组成 Pipeline 依次传输 Packet, 写入成功后, 返回写入 ack 信息.

7) DataNode 全部写入成功后 FSDataOutputStream 调用 close 方法, 关闭字节流. 读写模块检测到写入成功后, 发送信号给 Web 后端, 结束整个写入任务.

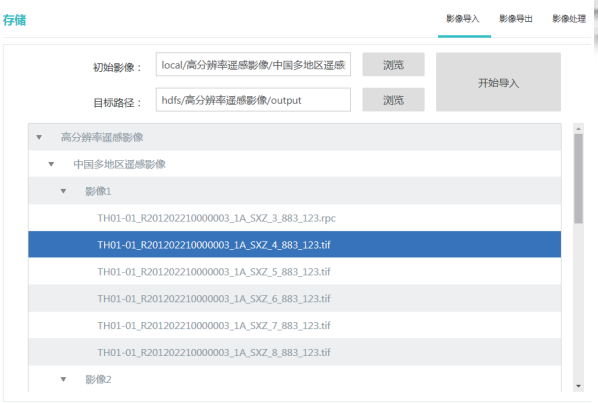

图 4 影像导入界面

UploadlFile 关键代码如下:

```
//初始路径
String FromPath = "/local /高分辨率遥感影像/中国多地区遥感影像/影像 1/ TH01-01_R201202210000003_1A_SXZ_4_883_123.tif";
//目标路径
String ToPath= "/hdfs/高分辨率遥感影像/output";
//程序配置
Configuration config = new Configuration();
config.set("fs.default.name","hdfs://master: 9000");
hdfs = FileSystem.get(new URI ("hdfs://master:9000"), config);
Path srcPath = new Path(FromPath);
Path destPath = new Path(ToPath);
boolean delSrc = true;
hdfs.copyFromLocalFile(delSrc, srcPath, destPath);
```

4.3 算法注册管理模块

如图 5 所示, 算法注册模块提供影像处理算法包注册和管理的功能. 由于 MySQL 数据库适合存储结构化数据, 同时千万级别以下查询响应延迟低, 所以将算法的名称、算法类型、算法包路径等元数据信息, 保存在 MySQL 数据库进行持久化. 算法包的数据量不适合使用 MySQL, 将分别写入三级缓存模块和 HDFS 模块.

图 5 算法注册界面

主要工作流程如下:

1) 用户在浏览器端, 填写算法名称, 算法版本, 算法简介, 从本地文件系统选择要注册的算法包.

2) 注册保存后, 后端将算法元数据信息发送给

© 中国科学院软件研究所 http://www.c-s-a.org.cn

MySQL 数据库,

3) 算法包信息同时发送给三级缓存模块和 HDFS 模块, 进行写入操作.

4) 使用算法包时候, 查询 MySQL 数据库表, 得到算法包路径信息.

5) 根据算法路径信息, 先向三级缓存模块查找, 如果查找失败, 最后调用 HDFS 的 Java 接口查找.

数据库表设计结构如表 1.

表 1 算法注册表结构

字段名称	字段类型	默认值	是否允许为空	索引	说明
ID	Int(20)	自增编号	否	主键	算法编号
Name	Vachar(20)	NULL	否	无	算法名称
Version	Vachar(20)	NULL	否	无	算法版本
Function	Vachar(20)	NULL	否	无	算法功能
Is_Supervised	Tinyint(1)	NULL	否	无	是否监督学习
Is_MapReduce	Tinyint(1)	NULL	否	无	是否并行处理
Location	Vachar(20)	NULL	否	无	算法路径名
Filename-extension	Vachar(20)	NULL	否	无	算法包缀名
Introducion	Text	NULL	是	无	算法简介

4.4 计算处理模块

计算处理模块提供影像处理的功能, 支持单机和分布式处理两种模式. 算法表结构中 Is_MapReduce 字段, 决定该算法的处理模式.

单机处理模式跟传统 ENVI, IMAGINE 等软件计算处理方式相同, 往往耗时较长. 由于很多影像处理算法可以结合 MapReduce 进行改进, 例如 K 均值, ISODATA[9–11]. 只要实现 Map 和 Reduce 两个函数, 就可以采用分布式处理模式, 如算法 2.

算法 2. 改进后的 K 均值算法

1. 在 Main 函数中, 读取初始 n 个中心位置, 保存到 Configuration 对象中.
2. 重写 Map 函数, 读取遥感信息图像, 计算每个像素点到 n 个中心位置的欧式距离, 距离最短的则为该像素点所属的聚类中心, 输出<聚类中心 id, 对应像素点>.
3. Shuffle 过程: 不同聚类中心, 经过哈希后, 哈希值相同的, 传给同一个 Reduce 节点.
4. 重写 Reduce 函数, 每个聚类中心点, 根据对应的所有像素点, 计算均值. 得到新的聚类中心点, 输出<聚类中心 id, 新的聚类中心 id>.
5. 计算新的聚类中心和之前聚类中心的距离, 如果小于设定阈值, 输出结果. 否则, 新的聚类中心 id, 写入 Configuration 中, 跳转到步骤 2.

基于上述算法, 该模块工作流程如下:

1) 计算模块接收到算法包元信息, 根据 Is_MapReduce 值, 若为真, 开启分布式处理模式.

2) 解析算法包信息, 开启多个 Map 任务, 产生的新的键值对写入本地三级缓存系统, 进行分区.

3) 新的键值对, 按照分区序号通过 Shuffle 过程, 发送给不同 Reduce 节点.

4) 不同 Reduce 节点, 按照算法包实现的 Reduce 函数, 产生新的键值对, 写入三级缓存系统, 输出结果.

5 系统性能测试

5.1 硬件环境

由 6 台普通 PC 机, 千兆网卡, 千兆交换机组成服务集群. 一个作为 NameNode 主节点或三级缓存模块的 Client 和 Master, 其它作为 DataNode 或三级缓存的 salve 节点. 其中主节点装载 MySQL 数据库, 版本为 5.7.20. 节点配置如表 2.

表 2 集群软件及硬件配置

名称	详细配置
操作系统	CentOS6.5
JAVA-JDK	1.8.0
Hadoop	2.6.5
CPU	Intel i5-8400
内存	2 GB
固态硬盘	128 GB
磁盘	500 GB

5.2 影像读取速度测试

测试条件采用 145 张 TIFF 格式的影像数据, 单幅大小约为 178 MB, 总容量为 25.81 GB. Hadoop 配置文件 hdfs-site.xml 参数设定如下:

```
<property>
<name>dfs.block.size</name>
<value>134217728</value> --块大小 128 M
<description>Block size</description>
```


© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
</property>
<property>
<name>dfs.replication</name>
<value>3</value> --副本数量为 3
<description> </property>
```

先直接使用 HDFS hadoop fs-get 命令, 进行读取测试. 再测试加入三级缓存模块后的读取速度测试. 每种测试方法反复进行 5 次, 记录耗费时间. 测试结果如图 6 所示.

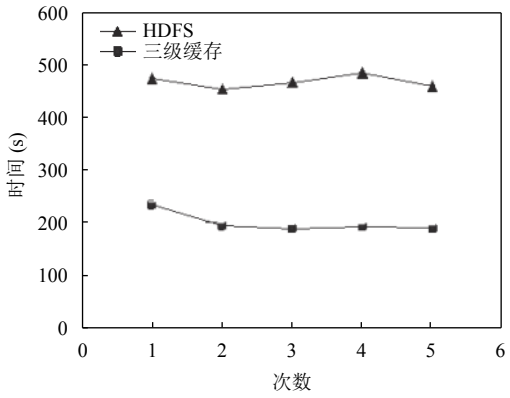

图 6 影像读取速度测试

根据测试数据, 计算出 HDFS 读取影像耗费时间平均为 466.72 s, 平均读取速度为 55.30 m/s. 加入三级缓存模块后, 平均耗费时间为 200.06 s, 平均读取速度为 129.01 m/s. 实验结果表明, 三级缓存模块可以将影像读取速度提升 2 倍以上, 有效解决影像数据读取缓慢的问题.

另外注意到, 三级缓存模块第一次读取耗费时间稍长, 之后就趋于平稳. 经过分析, 原因是首次读取时, 很多数据块可能位于底层次存储结构中. 再次读取时, 数据块热度值更新, 在缓存队列中的位置发生前移.

5.3 影像聚类处理测试

分别输入 178 MB、365 MB、712 MB、1424 MB 的影像数据, 模拟不同数据量对聚类时间的影响. 再使用单个节点, 模拟传统单节点聚类处理过程. 最后分别使用 2 个, 3 个, 4 个, 5 个计算节点, 模拟不同集群性能对处理时间的影响.

算法包中 Main 函数, 调用 JobConf 中 setNumReduce Task 方法, 将 Reduce 数量设置和计算节点数量相等. 测试结果如图 7 所示.

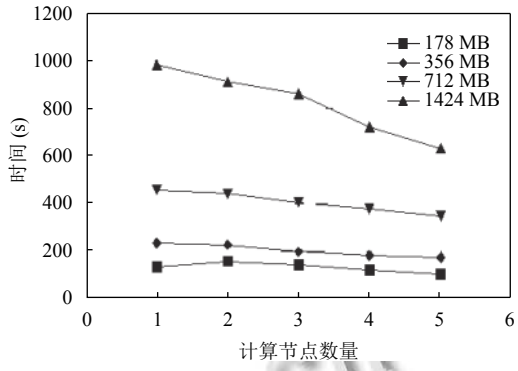

图 7 聚类处理时间测试

根据实验结果发现, 在数据量比较小的时候, 随着节点数量增加, 时间曲线下降不明显. 尤其是当数据规模只有 178 MB 情况下, 两个计算节点处理时间会稍许增加. 这是因为开启多节点进行聚类处理, 任务切割、网络通信时间会增加. 数据规模不大情况下, 反而导致整体时间开销增加.

随着输入数据量的增大, 可以发现时间曲线下降效果较为显著. 当达到 1 GB 数据规模时, 时间开销较传统单节点处理方式, 可以减少 40% 以上.

6 总结和展望

本文根据陕西某测绘所遇到的影像存储需求量大, 读写速度慢, 计算时间周期长, 多用户无法共享数据等问题. 基于 Hadoop 开源框架和 LFU 算法, 构建了一套影像业务管理系统, 提供影像存储, 算法注册, 影像处理功能. 实验结果证明, 该系统有效解决了上述问题, 其中三级缓存模块显著提升了整体系统的性能.

由于算法注册模块是通用模块, 同时支持基于单节点开发的算法包, 以及基于 MapReduce 计算框架开发的算法包. 之后工作, 可以就 ISODATA, Sobel 边缘检测[12]等常见的遥感影像处理算法如何结合 MapReduce 框架, 进行算法包的开发研究.

参考文献

1 冶鑫晨, 于炯, 钱育蓉. 基于 Hadoop 的遥感影像节能存储策略. 电子技术与软件工程, 2016, (8): 196.
2 郝树魁. Hadoop HDFS 和 MapReduce 架构浅析. 邮电设计技术, 2012, (7): 37–42. [doi: doi: 10.3969/j.issn.1007-3043.2012.07.008]
3 Borthakur D. HDFS architecture guide. The Apache Software Fundation, 2008.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

4 高见文, 薛行贵, 罗杰, 等. 基于迭代式 MapReducede 的海量数据并行聚类算法研究. 中国科技论文, 2016, 11(14): 1626–1631. [doi: 10.3969/j.issn.2095-2783.2016.14.014]
5 Elleuch W, Wali A, Alimi AM. An investigation of parallel road map inference from big GPS traces data. Procedia Computer Science, 2015, 53: 131–140. [doi: 10.1016/j.procs.2015.07.287]
6 Vaidya M. Critical study of performance parameters on distributed file systems using MapReduce. International Conference on Information Security and Privacy, 2015. [doi: 10.1016/j.procs.2016.02.037]
7 陈华, 陈书海, 张平, 等. K-means 算法在遥感分类中的应用. 红外与激光工程, 2000, 29(2): 26–30. [doi: 10.3969/j.issn.1007-2276.2000.02.008]
8 王磊, 孟昭鹏, 刘亚琼. 一种基于 LFU 置换的 BWT 压缩算法的改进. 微计算机应用, 2008, 29(3): 80–83. [doi: 10.3969/j.issn.2095-347X.2008.03.017]
9 Anandkrishna R, Kumar D. Improving mapreduce for incremental processing using map data storage. Procedia Computer Science, 2016, 87: 288–293. [doi: 10.1016/j.procs.2016.05.163]
10 Ling X, Yuan Y, Wang D, *et al*. Joint scheduling of MapReduce jobs with servers: Performance bounds and experiments. Journal of Parallel and Distributed Computing, 2016, 90–91: 52–66.
11 蒋利顺, 刘定生. 遥感图像 K-Means 并行算法研究. 遥感信息, 2008, (1): 27–30. [doi: 10.3969/j.issn.1000-3177.2008.01.006]
12 徐昌荣, 王聪颖. 基于 Hadoop 集群的 Sobel 边缘检测. 江西理工大学学报, 2013, (3): 38–41, 74.

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn