Hdspace 分布式机构知识库系统的小文件存储①

卞艺杰, 陈 超, 李亚冰, 陆小亮

(河海大学 南京, 211100)

摘 要: 机构知识库作为一种新型的学术交流模式和开放获取活动的绿色通道已逐渐成为国内外图书情报界关注的新焦点,随着机构库的发展其数据规模也在不断扩大, 传统的存储模式已经不能满足日益增长的存储需求. 在对机构库内容存储特点的研究基础上建立基于 HDFS 与 Dspace 的分布式机构库 Hdspace. 首先提出一种小文件合并生成新的存储文件, 并对文件提出基于学科分类的两级索引, 结合索引预缓存机制提高小文件的读取响应, 为海量小文件存储及后续的信息高效利用提供了一种解决方案, 通过模拟测试显示本模式能够大大提高机构知识库小文件的存储、读取以及检索效率.

关键词: 机构知识库; HDFS; 海量小文件; Dspace

Storage of Small Files in Hdspace Distributing Institutional Repository System

BIAN Yi-Jie, CHEN Chao, LI Ya-Bing, LU Xiao-Liang

(Business school of Hohai University, Nanjing 211100, China)

Abstract: The development of Institutional Repository requires a massive resource accumulation, the demand for storage keeps increasing especially for the small files. This article proposes a distributing storage model Hdspace which is based on Dapace and HDFS to resolve the problem of the storage of massive small files of Institutional Repository. First by a means of merging small document files to get new storage files, then by establishing two indexes based on subjects and index pre-caching to improve the file-reading response, finally put forward a method for the storage of massive small files.

Key words: institutional repository; HDFS; massive small files; dspace

机构知识库(Institutional Repository, 简称 IR), 又称学术库、机构资料库、机构仓储, 源自“学术交流”和“开放获取”两大驱动因素[1], 是学术机构为捕获并长期保存机构的知识成果而建立的数字资源库, 也是国际图书情报界近几年出现的一个新的应用领域和研究热点.

国外 IR 研究起步较早, 可以分为理论和技术研究两个方面. 理论研究主要集中在 IR 资源建设、开放获取等, 文献[2]研究了 IR 的内容建设并提出了相应的策略, 文献[3]提出运用 OAI-PMH 元数据收割技术方便、低成本地获取数据信息, 文献[4]提出一些提高作者自存储意识以丰富 IR 内容的途径. 技术研究主要是对 IR 软件平台的实践开发, 开源软件 DSpace、Eprints、Fedora 应用最为广泛.

国内 IR 研究起步晚于国外, 也可以分为理论和技术研究. 理论研究主要包括 IR 建设制度、政策和机制[5]、资源建设[6]、内容质量控制[7]、法律问题[8]等几个方面. 技术方面主要是对 Dspace 开源软件的应用实践与二次开发.

学术资源的存储是机构知识库的最主要功能, 在存储资源中文档类文件(如 Word、PDF、PPT 等)的数量约占 70%, 该类文件主要是已发表论文、技术报告、工作报告、论文预印本、会议报告和调查资料等[9], 特点是文件本身很小但需要全文检索. 随着机构知识库

① 收稿时间:2013-07-17;收到修改稿时间:2013-10-14

的发展, 这类小文件的更新速度和累积量都不断提升, 已经成为 IR 建设使用中亟待解决的问题.

HDFS(Hadoop distributed file system)是 Hadoop 框架下具有高度容错性的分布式文件系统, 已经被广泛应用于大规模数据的存储、分析中, 很多研究教育机构、公司等都已采用 HDFS 来解决快速增长的数据问题. HDFS 设计主要针对大文件的读写, 大量小文件的存储会降低 HDFS 文件系统整体性能, 不能被很好地应用在机构知识库等以小文件存储为主的系统中.

Hdspace 是基于 Dspace 和 HDFS 而实现的一种高效的分布式机构知识库系统, 针对以上问题进行优化改进, 能够解决机构知识库发展中的存储需求.

1　Hdspace存储模型

图 1 为 Hdspace 存储模型. 用户在 Hdspace 中提交资源后, 资源首先需要经过文件过滤器的筛选, 过滤条件包括文件大小、类型等, 通过筛选的文件统称为小文件. 随后对这些小文件提出一种合并策略, 一定数量的小文件合并后生成新的存储文件, 一般对属于同一学科的小文件进行合并. 在将新的存储文件写入系统的同时更新索引文件. Hdspace 中的索引包括两级, 一级索引是文件所属的资源集合, 如管理学、情报学等; 二级索引是具体的资源条目. 在需要读取文件时, 依次在一级索引和二级索引中查询, 缩小了查询范围, 能够保证较高的读取响应.

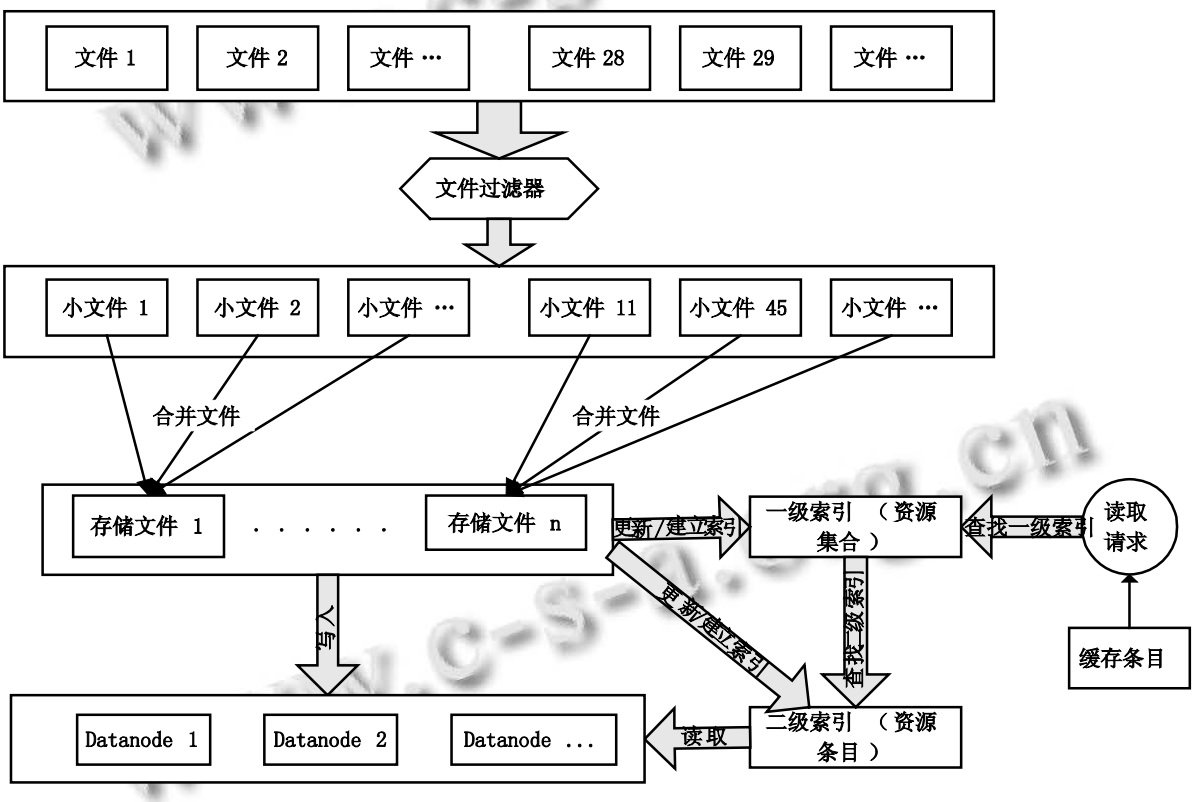

图 1　Hdspace 存储模型

2　Hdspace存储层设计思路

Hdspace 系统基于 Dspace 开源项目和 Hadoop HDFS 文件系统, 存储层参考 Dspace 进行设计. 图 1 Hdspace 存储模型展示了其存储层设计的核心: 首先对小文件进行合并生成存储文件, 再基于机构知识库的存储特征对合并后的文件建立二级索引, 通过索引预缓存提高文件读取的响应速度. 本节详细介绍存储层具体的设计思路.

2.1 基于小文件合并的存储文件生成策略

HDFS 将文件划分成一个个 block(块), 块的默认大小是 64M. HDFS 的命名空间持久化在一个叫做 FsImage 的文件中, HDFS 启动时由 NameNode 节点将其加载到内存中. 大量小文件会造成 NameNode 节点内存不足, 生成过大的 FsImage 文件降低读取文件时

文件的查找效率[10]. 对每一个文件的读写操作, HDFS 首先在命名空间中查询, 查找文件的块地址、文件大小等信息, 然后再在 DataNode 空间中进行检索. 当读取的文件很小时, 读写过程中主要时间都消耗在了检索查询中, 而不是文件数据的传输, 影响服务器集群的处理效率.

Hdspace 利用小文件合并来生成存储文件. 首先实现一个过滤器对文件按类型和大小进行过滤, 选择可以进行全文检索的文档类文件, 本文文件大小设定阀值为 10M, 当文件大于 10M 时则视为大文件, 不需要进行合并. 过滤后 Hdspace 按照文件条目所属资源集合为单位对过滤后的小文件进行合并成为文件块. 资源集合是具有一定相关性的资源条目的集合, 一个资源条目只属于一个资源集合. 通常集合按照学科范围、时间等划分, 机构知识库中的文件可以按照学科领域来划分. 新的文件块内资源条目具有很大的关联性, 在以后的数据处理中就可以将文件块分配给一个 MapReduce 任务, 避免了因任务的计算量太少而浪费任务分配和切换的时间, 减少数据在集群中的移动, 正符合 Hadoop 移动计算比移动数据更有效的原理.

2.2 建立二级索引优化文件读取速度

小文件合并后 NameNode 节点内存是整个文件系统的性能瓶颈, 因为所有的文件元数据信息需要存储在其内存中, 将小文件合并后可以减少文件的数量, 节省很多内存空间, 但是合并后的文件读取若使用 HDFS 提供的方法效率会很低. 现有研究通常为小文件建立索引, 在读取文件时通过索引定位文件存储的节点和合并后的文件位置以及其他元数据信息. 一些研究使用的是全局索引, 当小文件数量较多时, 索引文件过大, 查找效率明显下降. 假若有十万个小文件, 索引文件中的每条记录必须包含的字段包括文件名(128b)、块 ID(32b)、文件起始位置(32b)和文件长度(32b)四项元数据, 那么需要约 3.2G 内存空间, 文件数再多的话, 内存容量将明显不足.

Hdspace 系统采用两级索引来建立小文件元数据索引, 将大的索引文件以合理的规则划分为小的索引文件. 以资源集合为一级索引, 每个资源集合下的资源条目内容作为二级索引, 这样在查找的时候先根据资源条目所在集合进行查找, 再到相应的二级索引文件中进行查找. 虽然多了一个在一级索引中查找的过程, 但是由于资源集合数不会太多, 其查找时间是很小的, 经过划分的二级索引文件比全局索引文件小的多, 所以整体上会提高查找效率. 同时二级索引文件也并非全部加载入内存, 可根据内存使用情况并结合缓存策略进行灵活调度, 解决内存不足的问题.

2.3 基于索引预缓存的文件响应速度优化

这里提出的索引预缓存是指通过用户当前访问的数据预测用户下面将会访问的数据, 并将其索引调入缓存. 若能准确预测, 就可提前将用户将要访问的数据载入缓存, 当用户访问时就能得到较快的系统响应.

机构知识库中用户在下载/浏览资源条目前, 通常必须通过检索或是目录查找的方式得到“中间结果集”, 然后才能在其中选择需要的资源条目进一步访问. 在用户看到结果集页面与执行下载/浏览之间存在一个数秒的间隔, 在这段时间内通过提前缓存中间结果集中资源条目的索引, 在用户点击下载/浏览时就不用再执行一系列文件元数据查询, 直接进行传输文件即可, 这样可以很大程度上提高这些文件的请求响应. 这种响应提升并不需要太多的内存, 假设有 10 万个用户同时在执行检索, 每个结果集页面显示 20 条资源条目, 缓存一个文件的元数据信息需要 150B, 也只需要 0.3G 内存空间.

3 Hdspace存储层构架及关键技术实现

Hdspace 系统除了利用上述策略, 在实现时, 其存储层架构是系统的基础. Hdspace 存储层构建在 Hadoop 集群上的 HDFS 分布式存储系统上, 提供基本的文件保存和读取服务.

Hdspace 存储层的架构也是典型的三层结构设计: 用户接口层, 业务逻辑层和存储层[11], 而且为了提高性能建议, 采用将 Web 服务器和 HDFS 服务器集群分开的方式. 用户接口层即提供的用户界面, 用户通过该层提供的功能发送请求和接收反馈信息. 业务逻辑层是小文件读取和写入的功能实现层, 包括文件合并、索引构建和缓存构建等. 其具体架构如图 2 所示, 其中 W1、W2 等代表文件写入过程, R1、R2 等代表文件读取过程.

业务逻辑层包括文件合并、检索系统、小文件索引、缓存和 HDFS Client 等功能模块. 各模块具体功能及实现如下:

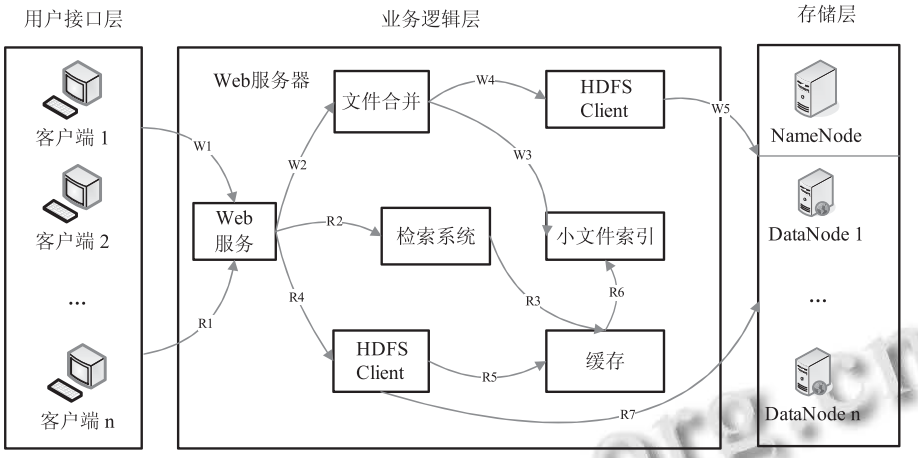

图 2　Hdspace 存储层整体架构

(1) 文件合并

文件合并功能包含 2 个阶段: 创建 SequenceFile 对象进行小文件进行合并. 通过过滤器的过滤, 对符合合并要求的文件进行合并, 首先根据资源条目所在的资源集合在一级索引中查找, 查找到资源集合对应的文件路径后, 创建 SequenceFile 对象, 并获得 SequenceFile 的 Writer 对象并对其进行配置, 准备写入文件. 在执行文件写入的同时开启一个新的线程, 将该资源条目对应的文件位值、长度等元数据信息写入资源条目二级索引. 资源条目写入成功后关闭输出流, 返回提交成功, 否则返回提交失败.

(2) 检索系统

提供文件检索功能, 该模块与存储模块无关, 但是本文基于“中间结果集”对 HDFS 进行读取优化, 需要依靠该模块, 所以在此列出.

(3) 小文件索引

构建小文件索引, 包括资源集合一级索引和资源条目二级索引, 提供索引文件创建、追加和删除记录等功能.

一级索引数据存储在关系数据库 PostgreSQL 中, 通过关系数据库访问接口提供访问, 使用 Java 中的 Map 数据结构保存. 因为资源集合在 DSpace 系统中已经存入数据库, 根据此索引只需要增加在资源条目添加的时候由系统生成值的字段, 所以可以保存在 PostgreSQL 中, 不影响处理效率. 一级索引中的数据类似于 Key/Value 结构, 可以使用 Java 中 Map 数据结构提高查询效率. 另外, 为保证检索效率, 必须在服务启动的时候根据数据库中内容初始化该 Map 对象并一直存在, 由于一级索引文件数不多, Map 对象占用内存很小, 所以系统开销很有限, 完全可以承受, 当有新的资源集合加入或有的被删除的时候, 需对该 Map 对象进行更新.

二级索引是通过开源项目 Lucene 创建的, 支持小文件元数据检索. Lucene 有一套完善的索引构建、更新和查找解决方案, 而且在索引文件小于 1G 时查询效率非常高, 可用于构建商用搜索引擎[12]. Hdspace 系统要创建的索引需要一些特殊的功能, 如每当用户添加资源条目的时候需实时更新索引文件; 多个用户在一个资源集合下同时添加资源条目时, 文件写入的并发控制; 压缩索引文件以减少内存占用等. 对于这些功能, Lucene 提供了完整的高效的解决方案, Dspace 系统的检索系统也是使用 Lucene 开发的. 所以这里使用 Lucene 对二级索引进行管理.

(4) 预缓存

为了更好地提升响应速度, 这里提供对用户感兴趣的“中间结果集”的缓存管理, 包括缓存空间维护, 缓存更新, 更新算法维护等功能.

在用户发出检索请求后, Web 服务根据用户检索条件查询符合用户需要的资源条目结果集, 返回给用户, 同时创建异步线程更新缓存, 在返回用户结果集到用户浏览结果集并确定点击下载/浏览操作之间的时间间隔内更新缓存内容. 当缓存模块接收到更新缓存内容请求时, 调用索引模块进行检索, 将当前结果集条目的元数据载入缓存. 当用户发送下载/浏览请求时, Web 服务调用 HDFS Client 在缓存中查找元数据开始读取数据并向客户端传输.

系统维护一个固定线程数量的线程池, 在每次接收到更新缓存请求的时候调用一个线程去处理, 若线程池内没有空闲线程则让该缓存任务等待. 这样可以将缓存更新任务占的系统资源维持在一个合理的范围内, 不影响系统整体性能. 本文选择 FIFO 算法实现缓存模块调度功能, 以最高效的方式淘汰最久以前的缓存条目. 具体实现是: 建立缓存池, 配置缓存池大小, 默认为 32M, 可以保存 20 万条文件元数据信息. 缓存池里面存储的是一个个 key/value 对, 文件名作为 key, 文件的 DataNodeID、起始位置和长度合起来作为 value. 该缓存池提供两个操作 put 和 get. put 往缓存池放入数据, 如果缓存池里面已有的数据达到了上限, 则根据缓存替换算法替换相应的数据, 如果还有空间直接放入就行. Get 操作根据 key 值获取相应的 value 值, 如没有则返回空.

(5) HDFS Client

HDFS Client 是 HDFS 文件系统的客户端, 其封装了操作 HDFS 文件系统与外界交互的 API, 包括读写文件和查询文件位置等. 当 HDFS 文件系统接收到文件读取请求时, 首先经过文件过滤器进行判断, 属于被合并了的文件则首先在缓存中查找文件的元数据信息, 若不存在, 则在索引文件中进行查找, 若还是查找不到则与 NameNode 通信. 查找到文件元数据后构建 SequenceFile 对象, 然后获得 SequenceFile 的 Reader 对象向 DataNode 发送读取请求, 将数据传输给用户后关闭输入流, 返回完成.

如图 2 所示, 用户有两种请求方式, 一种是提交文件的写入请求(W1), 一种是查询、浏览或获取资源的读取请求(R1).

(1) 文件写入

当 Web 服务器接收到用户提交资源请求时(W1), 首先判断是否需要做小文件合并, 若需要, 则进行文件合并(W2), 不需要的话直接使用 HDFS 写入接口进行写入即可. 文件合并后通过 HDFS 客户端准备将文件写入 HDFS 文件系统(W4), 在 HDFS Client 写入文件(W5)的同时, 调用小文件索引更新模块(W3)执行小文件索引及更新, 因为 Web 服务器主机和服务器集群是分离的, W3 和 W5 可以通过不同的线程同时执行, 彼此没有影响. 当 HDFS 写入成功后 Web 服务向客户端返回提交成功信息.

(2) 文件读取

在用户需浏览文件详细内容或下载文件时发送文件读取请求, 该请求频次高, 耗费系统资源最多. 当 Web 服务器接收到用户的读取请求时(R1), 首先通过检索系统根据用户提交的条件进行检索, 得到用户需要的资源条目结果集返回给用户浏览(R2), 同时将结果集中显示在用户界面中第一页的条目集合(默认 20 条)发送给缓存模块, 并开启一个单独的线程更新缓存(R3), 当用户浏览完返回的结果集页面请求下载或浏览详细时, Web 服务调用 HDFS 客户端准备读取文件内容(R4), HDFS 客户端首先在缓存中查找文件位置信息(R5), 没有则再到小文件索引中查找(R6), 查找到位置信息后则直接到 DataNode 节点读取数据, 返回给用户.

4 性能分析

本文使用 Cygwin 在 Windows 环境下模拟 Linux 环境, 在两台内存 4G Intel 奔腾 4 处理器的机器上进行 Hadoop 分布式环境搭建, 使用 Hadoop-0.20.1 版本的源码, 采用 Eclipse3.0 作为开发工具, 建立了一个 Hdspcae 的实验系统.

4.1 计算能力

本文选择 1000 个文本小文件(KB 级别)进行全文索引构建测试, 分别对 10、200、500 和 1000 个小文件建立全文索引, 分别记录了五次执行时间, 再取平均执行时间进行比较, 具体数据如表 1 所示, 其中 s 表示秒, m 表示分钟.

表 1 未合并和合并后的平均执行时间

对比项 \ 文件数	10	200	500	1000
未合并小文件	5.2s	10m	48m	135m
文件合并后	4.9s	34.5s	1m	3m

由上表可见, 当小文件个数很少时, 处理效率差别不大, 而当小文件很多时, 处理效率差别明显, 在本文两台主机的环境下, 500 个文件已经算很多了, 所以未合并时效率明显下降. 其原因是在未进行小文件合并时, 在对大量小文件进行处理, 需要创建大量的 Map 任务, 每个任务处理的内容很少, 但是每个 Map 任务的创建、调度和销毁相对于任务处理所耗费的时间是很多的. 而在进行文件合并后, Map 任务是根据合并后的文件大小进行分配的, 每个块默认为 64M, 500 个小文件合并后占据两个块, 所以只需要两个 Map 任务, 1000 个小文件只需要至多 5 个 Map 任务, 而对于每个任务的处理就是对流文件的读取和解析, 这

些对于 HDFS 系统来说都是非常高效的, 所以进行合并后, 文件的处理效率就不会因小文件数的增加而变的很低.

4.2 读取响应效率

本文选择 10000 个小文件进行读取效率测试, 每个小文件大小在 50k 到 300k 之间, 使用自动读取程序进行了 5 次读取操作, 每次读操作都模拟用户先检索后下载的方式先向缓存模块发送更新缓存请求, 然后隔四秒钟之后再对本次请求中的 3 个文件进行读取. 实验时连续发送 1000 个缓存任务, 每次缓存任务发送 10 个条目, 随机读取其中的三个文件, 最终读取 3000 个文件约 500M 大小. 记录 5 次的文件读取时间如表 2 所示.

表 2 优化前后效率对比表

	第 1 次 (秒)	第 2 次 (秒)	第 3 次 (秒)	第 4 次 (秒)	第 5 次 (秒)
未合并小文件	58.45	61.8	66.45	56.2	58.65
合并但未优化	51.45	55.8	60.45	49.2	52.65
合并并且优化	12.6	11.7	11.55	13.35	11.4

由上表可见, 在进行小文件合并之后, 文件的读取效率有所提升, 而在进行两级索引和缓存预加载优化后, 文件的读取效率又有较大提升. HDFS 在处理大量小文件时性能不佳, 主要是因为小文件过多时 NameNode 的文件查找时间增加, 而且相对于文件流的读取, 文件查找时间过大. 本文在进行文件合并后, 减少了 HDFS 中的文件数量, 减少了部分的文件查找时间, 所以在合并后读取效率有所提升, 但是提升的不多, 因为还是有很多查找需要 NameNode 执行. 而在进行两级索引和缓存预加载优化后, 当用户发送下载请求时, 直接从缓存中得到文件的位置信息进行文件读取即可, 大大减少了文件查找时间, 使读取性能得到很大提高.

5 结语

Hdspace 基于 HDFS 平台、Dspace 框架提出了一种海量小文件存储的综合解决方案, 有效提高了 HDFS 的小文件响应及读取效率, 不仅可以应用在机构知识库中, 也可以应用到具有相似存储特点的系统中. 另外 Hadoop 架构下还有 Hbase、Hive 等分布式处理技术[13], 这些技术在处理大量数据时很高效, 已经在一些领域应用. 当数据存储的内容日益增多达到很大存量后, 进行数据统计、知识挖掘等高级应用的时候, 很需要使用这些技术, 这些技术的集成应用有待进一步研究.

参考文献

1 Wang XM. Institutional repositories-background, issues and strategic considerations, http://lib.bjtu.edu.cn/pub/bjtu/xswhjl/hyjlzl/P020061012306324685509.ppt. 2009-8-17.

2 Foster NF, Gibbons S. Understanding faculty to improve content recruitment for institutional repositories. D-Lib, 2005,(l):31–37.

3 Hunter P. Metadata for harvesting. The Electronic Library, 2004,(2).

4 夏明春.机构知识库发展现状、问题及对策研究.图书情报工作,2008,(4):108–110.

5 张晓林.机构知识库的政策、功能和支撑机制分析.图书情报工作,2008,(1):23–27,19.

6 都平平.机构仓储的自存储和强制存储策略研究.图书馆杂志,2008,(9):15–18.

7 蔡迎春.机构知识库:基于开放存取的学术交流机制.情报理论与实践,2008,(5):680–683.

8 符玉霜.电子书的版权问题研究.现代情报,2011,(01):29–31.

9 郎庆华.机构知识库长期保存的策略分析.情报理论与实践,2010,(5):47–51, 62.

10 徐文强.基于 HDFS 的云存储系统研究[学位论文].上海:上海交通大学, 2011.

11 朱丽雪.基于 Dspace 的机构知识库构建[学位论文].天津:天津师范大学, 2010.

12 闻峥.基于 Lucene 的搜索引擎优化[学位论文].北京:北京交通大学, 2011.

13 陆嘉恒. Hadoop 实战.北京:机械工业出版社,2011.