HDFS中海量小文件合并与预取优化方法的研究

郑 通 郭卫斌 范贵生
(华东理工大学信息科学与工程学院 上海 200237)

摘 要 HDFS在存储海量文件时具有明显的优势,但在存储小文件占绝大多数的海量文件时,HDFS单个NameNode的存储架构会导致其性能严重降低。为此,提出一种基于合并思想的方案,即将小文件合并为大文件,同时建立小文件到合并文件的映射关系,并将其存于HBase中。为了提高读取速度,建立了基于LRU的预取机制。实验表明,该方法能明显提高HDFS在处理海量文件时的整体性能。

关键词 HDFS,海量文件,合并,映射,LRU,预取机制
中图法分类号 TP302.1 **文献标识码** A

Research on Optimization Method of Merging and Prefetching for Massive Small Files in HDFS

ZHENG Tong GUO Wei-bin FAN Gui-sheng
(School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China)

Abstract HDFS has a significant advantage on storing the massive files, however, its storage architecture which has only one NameNode will result in the decrease of performance when HDFS is used to store massive files which is mainly composed by small files. A solution based on the idea of that small files were merged into large files was proposed. Meanwhile, the mapping relationship from small files to merging files was established and stored into HBase. Finally, we provided a LRU based prefetching mechanism to improve the reading speed. The experiments show that the proposed method can improve the overall performance of HDFS with the large amounts of small files.

Keywords HDFS, Massive files, Merging, Mapping, LRU, Prefetch mechanism

1 引言

网络信息服务迅速发展,互联网中各种类型的数据也在急剧增长。为了存储日益增长的数据,Google File System[1],Hadoop Distribute File System[2],Lustre[3]等分布式存储文件系统应运而生。其中,HDFS因其高速性、低成本、高扩展性、高灵活性以及强容错能力,被广泛应用于海量数据存储领域。一次存储多次读取是HDFS的设计原则,其不但可以存储不断产生的海量文件,而且具有快速读取文件的性能。包含HDFS的Hadoop还包含高性能的分布式计算平台MapReduce[4],与HDFS紧密结合,分别用来处理与存储海量数据。HDFS最初是为存储超大文件而设计的,适合存储并访问具有高吞吐率的数据。研究发现,Facebook、Twitter、微博、QQ等社交软件每天产生的社交数据总量都在PB级别,而这些数据大多是一些kB级别的小文件,包含日志、照片、短消息等[5]。所谓小文件,就是其大小远小于HDFS的存储块的大小(默认64MB),当HDFS用来存储海量小文件时,其性能会显著降低。

HDFS是一个主从式架构,由一个NameNode(Master)节点与多个DataNode(Slave)节点组成。NameNode主要用于存储管理HDFS的命名空间,DataNode用于存储文件的数据块。HDFS的架构如图1所示。

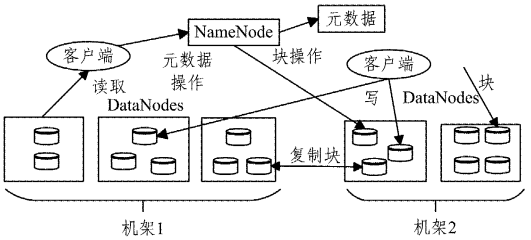

图1 HDFS架构

HDFS中的每一个文件都在NameNode的内存中存储一条关于该文件的元数据以及块的元数据。文件元数据包括文件名、大小、权限修改时间等,块的元数据包括该文件所对应数据块的列表以及数据块所存在的DataNode位置等[6]。无论文件的大小,每个文件的元数据大小均为150字节左右。因此,当HDFS存储的海量文件中小文件占大多数时,NameNode中会存储大量的元数据,内存资源被大量消耗,导致HDFS的性能显著降低,并且可存储文件数量受到NameNode内存大小的限制。当要读取的文件中小文件较多时,需要频繁地向NameNode发送获取元数据的请求,并在多个DataNode

本文受国家自然科学基金资助项目(61672227)资助。

郑 通(1991-),男,硕士生,CCF会员,主要研究方向为软件工程与云计算;郭卫斌(1968-),男,博士,副教授,CCF会员,主要研究方向为高性能计算、计算机应用与软件工程,E-mail:gweibin@ecust.edu.cn(通信作者);范贵生(1980-),男,博士,副教授,CCF会员,主要研究方向为软件工程及云计算。

之间跳跃寻找并读取小文件，大大降低了读取的性能。

另外，HDFS 不适合低延迟的访问。读写一个文件时经过的通信次数较多，在读写小文件时所占用的时间将远远少于为了读取文件所做的例如 RPC 通信、socket 建立连接以及寻址等所花费的时间。

基于以上问题，本文提出了一种缓解 NameNode 内存压力的方案，即将海量文件中的小文件合并为大文件，以减少文件数量，同时建立小文件到大文件的映射，以此检索合并文件中的小文件。另外，为了加快文件的读取效率，建立了文件的预取机制。实验证明，本方案在处理海量小文件存储时的效率有了显著的提高。

2　相关研究

目前针对海量文件中的小文件问题，Hadoop 自身提供了 3 种解决方案：Hadoop Archive[7]，Sequence File[8] 和 CombineFileInputFormat[9]。这些方法在解决了海量小文件占用 NameNode 大量内存的同时也带来了一些新的问题。利用 Hadoop Archive 技术合并小文件后，源文件不会自动删除，并且存在 Har 文件不支持压缩以及一旦创建就不可再修改等缺点。Sequence File 技术没有建立小文件到大文件的映射，查找一个小文件时需要遍历整个 Sequence File 文件，读取效率十分低下。

部分学者从不同角度对此进行了研究。游小荣、曹晟基于教育资源小文件间的关联关系提出了存储优化方案，即将小文件合并成大文件并建立索引机制和小文件预取机制来提高文件的读取效率[10]。Zhang Shuo，Miao Li 等人[11] 提出了根据文件大小范围来进行存储，用 HBase 来存储小于 50kB 的文件，50kB～1MB 的文件按照其提出的合并存储方法来存储，大于 1MB 的文件利用 AVRO 来存储。Patel A 与 Mehta M A[12] 提出一种将相关联的小文件合并为大文件，并建立预取与索引机制，但在读取小文件时需要提供小文件名与大文件名。赵晓永等提出一种基于 Hadoop 的海量 MP3 文件存储架构，利用 MP3 文件自身包含的丰富描述信息，通过归类算法将小文件归并到 Sequence File 中，同时通过引入索引机制，解决了 MP3 文件过多时引起的 NameNode 内存占用过多而导致的性能降低的问题[13]。淘宝、Facebook 针对自己平台产生的特定小文件，基于 HDFS 分别开发了 TFS：Taobao File System[14] 与 Haystack[15] 存储系统，以实现对小文件的高效存储。

以上方案均在一定程度上解决了海量文件中的小文件问题，但仍存在一些问题。例如，一些方案只针对某一领域的应用，无法应用到其他领域。还有一些改进方案在访问文件时既需要提供小文件名，又需要合并文件名，给文件的访问带来诸多不便。而本文所提方案只需提供小文件名，并且不只是针对某一单一领域的应用。

3　小文件处理方案

本文方案主要包括 3 个模块，即文件判断合并模块、建立索引模块和文件元数据预取模块。

为了减少小文件给 NameNode 带来的内存压力，本方案在文件上传时判断文件大小并分别进行处理。对于小于块大小的小文件，将多个小文件合并为一个合并文件，当合并文件的大小达到块的大小时便将其上传到 HDFS；对于大于块大小的大文件，直接将其上传到 HDFS。为了检索小文件，在合并小文件的过程中建立小文件到大文件的映射关系，即小文件在合并文件中的偏移量以及小文件的大小。将合并文件上传 HDFS 时，同时也将映射文件存储到 HBase。为了提高读取文件的效率，本方案基于 LRU 做了预取工作，即通过访问 HDFS 的审计日志，取出最近的 N 条访问记录，根据 LRU 算法，取出 n 条访问记录，结合 NameNode 中的元数据信息，将 n 个文件的元数据信息置于客户端内存中。当访问文件时，根据文件名以及 HBase 中的映射关系得到合并文件名时，即可到内存中查询是否存在该文件，若存在，则根据内存中的元数据信息直接向 DataNode 发送读请求，无需经过向 NameNode 建立 RPC 请求等步骤，从而提高了读取文件的速度。通过这 3 个部分的紧密结合，可以有效地改善 NameNode 的内存限制问题以及读写效率低的问题。本文方案的文件处理框架如图 2 所示。

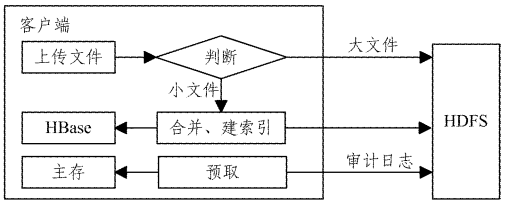

图 2　文件处理框架图

3.1　文件合并

客户端向 HDFS 上传文件时，首先对文件进行判断。若文件为小文件，则在客户端内存中建立一个临时文件，每接收一个小文件就将其合并到临时文件中，直至合并文件大小达到数据块的大小（默认 64MB）或者时间阈值 $T1$ 时，就将该合并文件上传到 HDFS。该合并文件的名字为向 HDFS 上传文件时的时间，例如 20170409102345。设定时间阈值 $T1$，是为了防止一些小文件已合并到合并文件中且合并文件未达到块的大小，后续一段时间内没有小文件上传或因网络、系统故障等原因接收不到其他小文件而导致临时文件一直处于内存中而不能及时上传到 HDFS 中，避免了要访问处于临时文件中的小文件而得不到访问结果。时间阈值 $T1$ 可以根据实际应用场景设定。实际情况下，临时文件的大小不可能恰好为默认块的大小，即当满足临时文件再多放一个文件就超过块的大小时，就将其上传到 HDFS。图 3 为填满小文件的临时文件结构图，其中空白区的大小小于下一个小文件的大小。

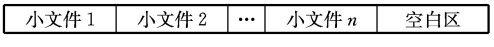

图 3　临时合并文件结构

若客户端上传的文件为大文件，则无需合并，直接将大文件上传到 HDFS 中，并建立一个临时索引文件来存储大文件名。

合并算法部分的伪代码如下所示：

```
WHILE DuringTime<T1
  IF NextFileSize>BLOCKSIZE
    THEN WriteToHDFS;
      AddBigFileIndex;
```

```
  ELSE IF TmpFile+NextSmallFileSize =<BLOCKSIZE
    THEN MergeSmallFile;
         AddSmallIndex;
  ELSE
    UploadTmpFile;
    StoreSmallIndex;
    MergeSmallFile;
    AddSmallIndex;
  END IF
END WHILE
```

3.2 映射关系

为了检索合并文件中的小文件，本方案在小文件合并的过程中建立索引文件，即每当合并一个小文件到临时文件后，便将此小文件名、小文件的大小以及其在合并文件中的偏移量存于索引文件中，当向 HDFS 上传临时文件时，将小文件的索引文件以及其对应的合并文件名存于 HBase 中。对于大文件，也建立一个记录文件，只记录其文件名，每隔 $T1$ 时间将其存于 HBase 表中。大文件的记录文件以及小文件的索引文件如图 4 所示。

小文件 1	小文件 2	小文件 3	…	大文件 n

(a)大文件的记录文件

小文件 1	文件大小	偏移量
小文件 2	文件大小	偏移量
⋮	⋮	⋮
小文件 n	文件大小	偏移量

(b)小文件的索引文件

图 4 文件索引文件

HBase 是一个非关系型、面向列存储的分布式数据库，它可以实现高性能的并发读写操作，从而高效地存取小文件[16]。HBase 数据表主要包括主键、列族以及时间戳。主键即 Row Key，它是检索记录的主键，HBase 中的每一个列都属于一个列族。

在本方案中，文件的名字作为 Row Key，即根据文件名创建一个 content 列族。列族中包含大文件标志、合并文件名、小文件大小和小文件在合并文件中的偏移量。若文件为大文件，则其 flag 为 1，其余 3 列为空；否则 flag 为 0，其余 3 列分别为小文件索引文件中的数值。HBase 中的映射结构如表 1 所列。

表 1 HBase 中的映射表结构

Row Key	Time Stamp	列表			
		大文件标志	合并文件名	小文件大小	偏移量
小文件 1	t_1	0	合并文件 1	100k	0
小文件 2	t_2	0	合并文件 1	97k	100
大文件 1	t_3	1	—	—	—
…	…	…	…	…	…
大文件 j	t_j	1	—	—	—
小文件 i	t_i	0	合并文件 j	…	…
…	…	…	…	…	…
小文件 n	t_n	0	合并文件 k	…	…

3.3 预取机制

当访问 HDFS 中的文件时，需通过建立 RPC 向 NameNode 发送获取文件元数据的请求，若请求访问大量的文件，海量高频的请求将给 NameNode 带来巨大的压力。为了减少 NameNode 的压力并提高海量文件中小文件的读取效率，本方案通过采用 LRU 算法，结合 HDFS 的审计日志，将一些最近访问的文件的元数据置于客户端内存中。

本方案通过周期性地分析 HDFS 的访问日志，即分析最新的 N 条访问记录，根据 LRU 算法，淘汰最长时间没有访问的记录，获得 n 个文件的名字，再结合 NameNode 中的元数据作进一步分析，获得它们存储位置的主机名以及块号，并将这 n 条索引信息存储于客户端内存中。客户端再次访问某个文件时，便可直接到内存中查找该文件的元数据信息。若存在，则无需再建立 RPC 请求，直接向 DataNode 发送读请求，从而加快了访问的速率。分析周期 $T2$ 根据系统需求设置。N 与 n 均不是固定数值，其大小取决于客户端内存以及特定应用场景。

3.4 读取过程

当客户端读取文件时，客户端首先根据文件名到 HBase 数据表中检索，若其对应的 flag 为 1，则该文件为大文件。若 flag 为 0，则该文件为小文件，获得其对应的合并文件的名称、大小以及偏移量。然后，根据大文件名或者合并文件名到内存中的预取索引表中检索是否有对应的文件元数据。若存在，则根据预取索引表中的主机名以及数据块号向指定的主机发送读请求，若为小文件，则结合 HBase 表中获取的小文件大小以及偏移量，便可读取指定小文件。若不存在，则通过原始 HDFS 的方法进行读取，即向 NameNode 发送 RPC 请求，请求获得文件的元数据，待 NameNode 返回元数据后再向 DataNode 发送读请求。读取过程如图 5 所示。

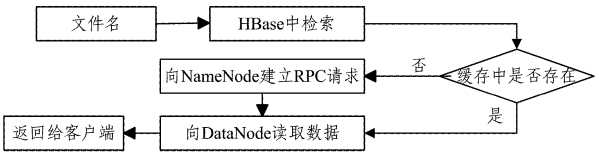

图 5 文件读取流程

根据读取过程可以得出，在命中时，客户端直接向 DataNode 发送读取数据的请求，节省了向 NameNode 发送 RPC 请求以及等待响应等步骤的时间花费，从而提高了文件读取的效率。

4 数值实验

4.1 软硬件环境

本实验在 4 台计算机搭建的 Hadoop 集群上进行，其中 1 台 NameNode 节点，3 台 DataNode 节点，数据块的副本数量为 3。计算机的硬件配置为英特尔酷睿 i7，处理器为双核 3.4GHz，内存为 DDR3 4GB，120GB 的硬盘。4 台计算机的软件配置如下：操作系统为 Ubuntu 14.04 64bit，jdk1.8.0，Hadoop 版本为 hadoop2.6.0，Zookeeper 版本为 zookeeper 3.4.5，HBase 版本为 hbase1.0.0。

本实验测试数据集为 100000 个文件，其中文件大小为 1～50kB的占 95%，64M～100MB 的文件占 5%，将 100000 个文件随机打乱，分 5 组测试数据，每组的数量分别为 20000，40000，60000，80000，100000。下面分别从写入速度、内存占用、读取速度 3 个方面进行实验。

4.2　写入速度实验

为了验证本文方案与原始 HDFS 在文件上传速度方面的差异，将 5 组数据分别通过改进方案与原始方法上传到 HDFS，并分别记录每组的所用时间。每组实验进行 3 次，取 3 次结果的平均值，实验结果图 6 所示。

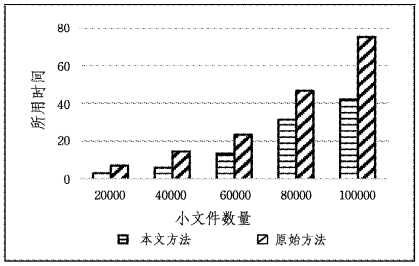

图 6　小文件写入时间对比

由实验结果可以得出，本文方案的上传文件速度优于原始 HDFS。因为客户端每次向 HDFS 上传文件都要向 NameNode 发送请求，由 NameNode 对客户端进行响应。而本文方案虽然在向 HDFS 上传文件时向 HBase 数据库写入索引信息占用了一些时间，但多个小文件合并成的合并文件上传时只需向 NameNode 发送一次请求，因此总体的写入速度仍然优于原始的 HDFS。另外，由实验结果可以看出，随着文件数量的增大，本文方案与原始方法的差距逐渐增大，优势更加明显。

4.3　内存占用实验

为了验证本文方案在元数据对 NameNode 内存占用方面的改进性能，将本文方案、HAR 方法以及原始 HDFS 的方法进行对比，每组文件上传之后记录 NameNode 的内存增长情况。每组实验进行 3 次，取 3 次结果的平均值，实验结果如图 7 所示。

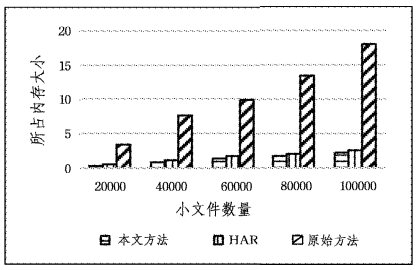

图 7　NameNode 内存占用情况对比

实验结果显示，在降低占用 NameNode 内存方面，本文方案与 HAR 均具有较好的性能，而且本文方法稍优于 HAR 方法，两者相比原始方法具有十分明显的优势，主要原因是本文方案通过合并小文件减少了 NameNode 中元数据的数量。

4.4　读取速度实验

为了验证本文方案在小文件读取速度方面的优势，将本文方案、HAR 方法以及原始 HDFS 的方法进行对比。通过本文方案将文件上传到 HDFS，每组上传后随机产生 5000 条访问日志与 5000 个访问的文件名，根据生成的文件名读取小文件。按照相同的步骤对 HAR 方法和原始的 HDFS 方法进行实验，每组实验进行 3 次，取 3 次结果的平均值，实验结果如图 8 所示。通过本文方案读取文件时需要使用预取模块的分析结果，在预取模块中，根据机器内存等情况，本实验中 N 取 2000，n 取 200，分析周期 $T2$ 取 5s。

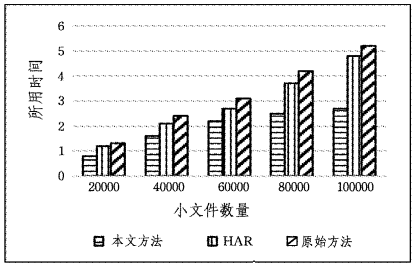

图 8　小文件读取时间对比

由实验结果可以看出，本文方案由于加入了基于 LRU 算法的预取机制，将一些文件的元数据置于客户端的内存中，减少了客户端向 NameNode 发送请求的次数，从而加快了海量文件的读取速度。另外，随着文件数量的增长，本文方案呈现出更加明显的优势。除此之外，当 $T2$ 取更合适的数值或者 n 取更大的值时，命中客户端内存中的文件的几率更大，从而使得本方案的优势更加明显。但 $T2$ 与 n 需要根据实际的应用以及物理设备进行配置，从而表现出更佳的性能。

结束语　本文通过分析 HDFS 的存储架构，针对 HDFS 在存储小文件占主体的海量文件时占用 NameNode 过多内存、存储文件数量受到限制以及访问小文件效率低等问题，提出了一种基于合并思想的改进方案，即将文件中的小文件合并为大文件，降低了 NameNode 中元数据的数量，减少了海量小文件对 NameNode 的内存占用。为了方便读取合并文件中的小文件，在合并文件的过程中同时建立小文件与合并文件之间的映射关系。同时，结合 LRU 算法建立的预取机制，加快了小文件的读取速度。实验结果表明，本方案在向 HDFS 写入文件、读取文件以及降低文件元数据对 NameNode 内存占用方面均有较为明显的改善，使 NameNode 不再成为系统性能的瓶颈，提高了 HDFS 在存取海量小文件时的性能。本方案未考虑小文件内容之间的关联性以及合并时的异常情况，接下来可以针对这两方面做进一步研究，使得小文件的读取效率更加高效。

参 考 文 献

[1] CHEN C T, HSU C C, WU J J, et al. GFS: A Distributed File System with Multi-source Data Access and Replication for Grid Computing[C]// International Conference on Advances in Grid & Pervasive Computing. Geneva: ACM, 2009: 119-130.

[2] 王意洁，孙伟东，周松，等. 云计算环境下的分布存储关键技术[J]. 软件学报，2012，23(4)：962-986.

[3] BRAAM P J. The Lustre storage architecture[EB/OJ]. http://www.lustre.org/documentation.

[4] 李建江，崔健，王聃，等. MapReduce 并行编程模型研究综述[J]. 电子学报，2011，39(11)：2635-2642.

[5] 李娜娜. 云计算平台下社交网络数据获取技术研究[D]. 北京：北京邮电大学，2013.

[6] 黄晓云. 基于 HDFS 的云存储服务系统研究[D]. 大连：大连海事大学，2010.

[7] Hadoop. Hadoop Archives Guide[EB/OL]. (2013-04-01)[2016-12-18]. http://hadoop.apache.org/docs/r1.2.1/Hadoop_archives.html.

（下转第 541 页）

务；根据服务的调用次数、注册时间对服务进行排序，并得到服务调用排行榜。

点击服务头像，可以查询服务信息，并提供申请使用入口，界面如图 6 所示。

图 6　服务简介

5.5　用户管理

平台提供服务商和开发商登录、注册入口。注册时，用户只需要填写邮箱、密码即可；点击注册按钮后，平台会发送验证邮件，用户点击邮件中的链接，打开用户完善信息界面。在完善信息界面，用户需要填写用户名、联系人、联系电话、身份信息等，平台管理员对用户信息进行审核。用户可以用用户名、邮箱、手机号在登录入口进行登录。

用户信息存储在用户信息表中，根据 type 字段区分管理员、开发商、服务商。Status 字段用来标识用户的状态：0 表示用户信息等待审核，在该状态下，用户可以登录平台，但是不能注册服务、创建凭证；1 表示用户已经通过审核，可以使用平台功能；－1 表示该用户未通过审核，不能登录平台。

5.6　统一访问地址

用户通过平台看到的 API 地址是统一网关入口的地址，界面如图 7 所示。

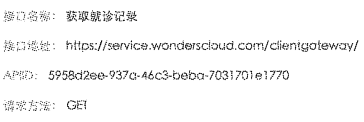

图 7　API 统一访问地址

开发商在应用程序中调用服务的过程分为两步：1）使用凭证的 app_ID＋app_KEY＋timestamp（当前时间戳）＋app_SIGNATURE，通过 sh1 加密作为参数向统一网关发送获取令牌（gettoken）请求，网关会返回令牌（token），令牌的有效期为 7200 秒；2）使用上一步获取的 token，apiid 以及 api 的所需参数向统一网关发送请求，网关接收到请求通过鉴权后，转发请求，并返回真实 API 的返回结果。

统一访问地址、网关鉴权极大地增强了平台的安全性，也简化了开发商的大量配置工作。

结束语　本文对基于 MEAN＋SpringMVC 的服务管理平台的使用场景、关键技术、设计方案进行了详细的阐述，并对核心功能进行了解释。基于此平台，既实现了对大量 API 高效、便捷的管理，也完成了 API 的汇聚共享，很好地解决了传统 API 管理工作中的问题。借助于统一网关，实现服务的统一调用、统一鉴权，极大地增强了 API 调用的安全性，从很大程度上减轻了开发者的工作负担。本文的重点是介绍服务管理平台的设计与实现，对统一网关、调用鉴权、消息通讯只进行了简单介绍，设计也相对简略。

参 考 文 献

[1]　百度百科[EB/OL]. http://baike.baidu.com.

[2]　卫培培，王建. 基于 BootStrap 的销售类网站设计[J]. 信息通信，2017(1)：91-92.

[3]　王云瑜，黄焯，龚家耿，等. 基于 Bootstrap 的学生论文管理系统的设计与实现[J]. 福建电脑，2015(7)：95-96.

[4]　MEAN 实践——LAMP 新时代替代方案[OL]. http://www.codesec.net/view/183734.html.

[5]　黄扬子. 基于 NodeJS 平台搭建 REST 风格 Web 服务[J]. 无线互联科技，2015(16)：57-59.

[6]　王越. 基于 nodejs 的微博系统的设计与实现[D]. 成都：电子科技大学，2014.

[7]　万里晴，杨浩. 探究基于 V8 引擎的 Node.js 在各应用领域的发展[J]. 通讯世界，2015(13)：97.

[8]　程桂花，沈炜，何松林，等. Node.js 中 Express 框架路由机制的研究[J]. 工业控制计算机，2016(8)：101-102.

[9]　雷德龙. 基于 MongoDB 的矢量空间数据云存储与处理系统[D]. 成都：电子科技大学，2014.

[10]　ARTHUR J，AZADEGAN S. Spring Framework for Rapid Open Source J2EE Web Application Developments：A Case SUidy[C]// International Conference on Software Engineeringj Artificial Intelligence，NETWORKING and Parallel/distributed Computing，2005 and First Acis International Workshop on Self-Assembling Wireless Networks. 2005：90-95.

（上接第 519 页）

[8]　Hadoop. Sequence File Wiki [EB/OL]. (2009-09-05)[2016-12-18]. https://wiki.apache.org /hadoop/SequenceFile.

[9]　DIEM C. Combine File Input Format[EB/OL]. (2013-09-22)[2016-12-18]. http://www.idryman.org/blog/2013/09/22/process-small-files-on-hadoop-using-combinefile-inputformat-1.

[10]　游小容，曹晟. 海量教育资源中小文件的存储研究[J]. 计算机科学，2015，42(10)：76-80.

[11]　ZHANG S，MIAO L，ZHANG D F，et al. A Strategy to Deal with Mass Small Files in HDFS[C]// International Conference on Intelligent Human-Machine Systems and Cybernetics. New York：IEEE，2014：331-334.

[12]　PATEL A，MEHTA M A. A Novel Approach for Efficient Handling of Small Files in HDFS[C]// IEEE International Advance Computing Conference. New York：IEEE，2015：1258-1262.

[13]　赵晓永，杨扬，孙莉莉，等. 基于 Hadoop 的海量 MP3 文件存储架构[J]. 计算机应用，2012，32(6)：1724-1726.

[14]　赵洋. 淘宝 TFS 深度剖析[J]. 数字化用户，2013(3)：58-59.

[15]　BEAVER D，KUMAR S，LI H C，et al. Finding a needle in Haystack Facebook's photo storage[C]// Usenix Symposium on Operating Systems Design and Implementation. New York：ACM，2010：47-60.

[16]　GEOGE L. HBase 权威指南[M]. 代志远，刘佳，蒋杰，译. 北京：人民邮电出版社，2013：302-304.