海量教育资源中小文件的存储研究

游小容 曹 晟
(电子科技大学计算机科学与工程学院 成都 611731)

摘 要 Hadoop作为成熟的分布式云平台,能提供可靠高效的存储服务,常用来解决大文件的存储问题,但在处理海量小文件时效率显著降低。提出了基于Hadoop的海量教育资源中小文件的存储优化方案,即利用教育资源小文件间的关联关系,将小文件合并成大文件以减少文件数量,并用索引机制访问小文件及元数据缓存和关联小文件预取机制来提高文件的读取效率。实验证明,以上方法提高了Hadoop文件系统对小文件的存取效率。

关键词 Hadoop,海量小文件,小文件合并,预取缓存
中图法分类号 TP302.1 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.10.017

Storage Research of Small Files in Massive Education Resource

YOU Xiao-rong CAO Sheng
(School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China)

Abstract As a distributed cloud platform, Hadoop is one of the most widely used cloud storage technology for applications with large datasets to provide reliable and efficient storage service, but it suffers a performance penalty with increased number of small files. In order to improve the efficiency of storing and accessing the small files on Hadoop, we proposed a scheme, based on the relationship of small files. In the scheme, a set of correlated files is combined into a large file to reduce the file count, indexing mechanism is used to access small file and metadata cache, and associated small file prefetching mechanism is used to improve the efficiency of file read. The experimental results indicate that the above methods can improve the storage and access efficiency of small file on Hadoop.

Keywords Hadoop, Massive small files, Merged small files, Prefetching and cache

1 引言

Hadoop是一种分析和处理大数据的软件平台,很擅长存储大量的半结构化的数据集。在此平台上,数据可以随机存放,所以一个磁盘的失败并不会带来数据丢失。Hadoop也非常擅长分布式计算,可快速地跨多台机器处理大型数据集合,它的核心设计包括分布式文件系统HDFS(Hadoop Distributed File System)和MapReduce。其中HDFS为海量的数据提供存储,MapReduce为海量的数据提供计算[1]。HDFS有高容错性的特点,它提供高吞吐量来访问应用程序的数据,适合那些有着超大数据集的应用程序,HDFS可以以流的形式访问文件系统中的数据。HDFS的架构是基于一组特定的节点构建的,这些节点包括Namenode(仅一个),它在HDFS内部提供元数据服务;Datanode(若干个),它为HDFS提供存储块。由于仅存在一个Namenode,因此容易造成HDFS的单点失败问题[2]。

本文所提到的教育资源小文件包括各种形式的教育资源,如word文档、pdf文档、ppt课件及文本资料等,只要与教育资源相关且大小远小于Hadoop的存储块大小64MB的文件都可称为教育资源小文件。教育资源小文件具有以下3个特点:1)这些文件的大小通常为几十到几百kB,与Hadoop的存储块大小64MB相比较小。2)小文件之间有关联性,因为知识的学习是循序渐近的,只有掌握了基础的知识才能学好新的知识,文中提到的关联关系即知识点的前驱后继关系。例如:要学习数据结构中二叉树的存储结构知识点,必须先学习二叉树知识点,了解什么是二叉树,二叉树有什么性质;要学习二叉树就要先学习树结构知识点,了解树是什么样的数据结构,树的表示方法,树的遍历方法有哪些等。在这个例子中二叉树的前驱是树结构,后继是二叉树的存储结构,通过知识点的层层递进不断地学习新的知识点。3)小文件的数量很大,网络中存在的各类学习资源量非常大。

针对海量教育资源的特点,将小文件直接存储在Hadoop的文件系统上存在一些问题[3]。Hadoop的文件系统HDFS是为处理海量大文件而设计的,在处理海量小文件时存在以下几点问题:1)因为HDFS只有一个Namenode节点处理文件的元数据信息,当直接存储海量小文件时会产生相应数量的元数据信息,这将快速耗费掉Namenode的内存,从而造成Namenode瓶颈问题;2)海量小文件直接存在HDFS

到稿日期:2014-10-21 返修日期:2015-01-07 本文受教育部——中国移动科研基金项目:海量教育资源去存储与获取关键技术研究与实现(MCM 20121041)资助。

游小容(1990—),女,硕士生,主要研究方向为云存储研究,E-mail:youxiaorong1218@163.com;曹 晟(1981—),男,副教授,硕士生导师,主要研究方向为云计算、网络安全。

上时，每次读小文件都要访问 Namenode 获取相关的元数据信息，这将严重降低 I/O 性能；3）HDFS 是随机存储文件的，对于文中提及的关联小文件，HDFS 没有考虑其之间的相关性；4）HDFS 是为大文件而设计的，但其并没有明确指出大小文件的分界点等。

2 相关工作

针对 HDFS 存储小文件存在的问题，目前已有的解决方案如下。

方案 1　Bo Dong 等基于 BlueSky 在线教育资源分享系统的应用，提出了一种新的改善 Hadoop 中小文件存储和访问效率的方法[4]。其中主要提出了两个基本思路：1）将相关的小文件合并成一个大文件，从而减轻名称节点的内存压力，提高小文件的存储效率；2）通过索引文件和数据文件的预取机制提高小文件的读取效率。

方案 2　李寅提出了采用分布式名称节点模型来处理 HDFS 的单节点瓶颈及扩展性问题[5]；提出了一种二级元数据分布算法，由目录信息和数据位置信息组成元数据信息，采用不同的分布策略进行分布式存储。该方法在保证高效率的元数据服务的基础上，考虑了元数据的负载均衡性和扩展性，并提出名称节点集群中元数据的可靠性机制。

方案 3　赵晓永等人提出了一种基于 Hadoop 的海量 MP3 文件存储架构[6]。该架构利用 MP3 文件自身包含的描述信息，通过在预处理模块中使用归类算法，将相关性强的文件合并为序列文件，以大量减少文件数量；同时引入高效的扩展一级索引机制，来快速定位到 MP3 文件所在的序列文件及其偏移位置；另外在富元数据管理模块中将 MP3 文件的富元信息进行集中索引和管理，从而解决 Hadoop 处理小文件时名称节点的内存瓶颈问题。

方案 4　Songling Fu 等针对分布式文件系统中海量小文件访问时的数据服务器优化问题[7]，提出了一种采用扁平式数据存储方法的轻量级文件系统 FlatLFS，用其取代传统文件系统对上层分布式文件系统来提供数据存储和访问支持，提高了数据服务器处理小数据块时的 I/O 性能，从而提升了整个分布式文件系统的性能。

综上所述，可以看出前 3 个方案通过不同的方法将小文件合并成大文件后存储在 HDFS 上，以减少文件的元数据信息，从而解决上节提到的问题 1）和问题 2）。方案 4 提出一个扁平式的存储系统来提高数据服务器处理小数据块时的 I/O 性能。这些方案虽然在一定程度上解决了 HDFS 存储小文件时 Namenode 的瓶颈问题，但是没有指明什么是关联文件，它们之间有什么关联关系。

3 方案描述

本文提出了一种基于 Hadoop 的海量教育资源小文件的存储优化方案，即利用教育资源小文件间的前驱后继关系，把同一课程的关联小文件归类合并成大文件，并对每个大文件建立局部索引，局部索引文件放在大文件的起始位置作为大文件的一部分，在上传大文件时一起存储于 HDFS 上。合并后，Namenode 只需要服务一个大文件而不是大量的小文件，所以可以大量减少元数据信息的数量，节省 Namenode 的主存空间，缓解 Namenode 的瓶颈问题。利用元数据缓存机制可减少客户端服务器与 Namenode 的交互次数，并利用关联小文件预取机制减少客户端服务器与 Datenode 的交互次数，通过减少 I/O 交互次数，提高小文件的读取效率；同时，利用碎片回收机制提高 HDFS 存储空间的利用率。

3.1 文件的合并与上传

考虑到小文件自身的特点，如果将小文件直接存于 HDFS，会快速耗费掉 Namenode 的存储空间，造成瓶颈问题，严重影响 HDFS 的性能。所以客户端发送写小文件请求到 Namenode 节点之前，必须先对小文件进行预处理，即关联小文件的合并操作[8]。小文件的关联是利用向量空间模型 VSM 算法来实现的。将小文件转换为相应的文本特征向量，并用式(1)计算其余弦夹角值，当计算出的余弦夹角值小于特定的阈值 F 时，就认为这两个小文件之间存在关联。

$$\cos(d_i,c_j)=\frac{\sum_{k=1}^{m} w_{ik} * w_{jk}}{\left[\left(\sum_{k=1}^{m} w_{ik}^2\right)\left(\sum_{k=1}^{m} w_{jk}^2\right)\right]^{\frac{1}{2}}} \tag{1}$$

小文件的合并过程需要在客户端与名称节点交互之前完成，本方法采用多个客户端服务器处理来自其它客户端上传的小文件[9]；然后再由这些客户端服务器向 Namenode 发起文件写请求，得到可写入的 Datanode 信息和数据块信息后，将合并后的大文件写入相应的 Datanode 块中。小文件的合并及交互过程由 4 个部分组成，分别为客户端、客户端服务器、名称节点 Namenode 和数据节点 Datanode。小文件合并交互结构如图 1 所示。

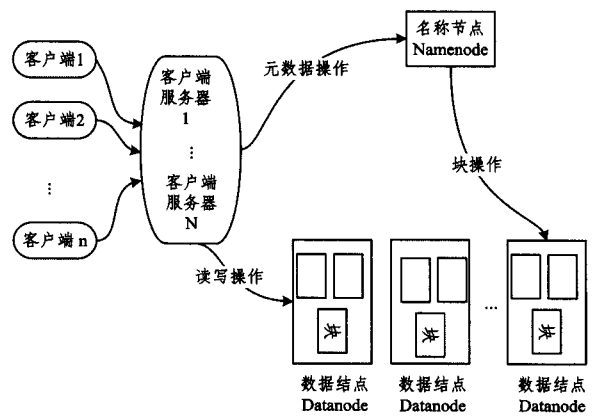

图 1　小文件合并交互结构

小文件的合并上传过程如下：客户端将需要存储的各类教育资源发送给客户端服务器，由客户端服务器通过式(1)计算出两个小文件之间文本特征向量的余弦值，当该值小于特定的阈值 F 时，就将其合并成一个大文件，直到大文件的大小达到 64MB 时，向文件系统写入文件。首先，客户端服务器从自己缓存的 Datanode 列表里选择访问最多的Datanode进行直接访问。如果没有，则向 Namenode 端发送文件写入请求，请求获得可以写入文件的 Datanode 信息。Namenode 为请求文件创建并保存相应的元数据信息，并将元数据信息返回给客户端服务器。客户端服务器得到可以写入的 Datanode 信息后，与 Datanode 端交互请求写入文件，Datanode 为其分配可写入的数据块，如果写入失败，Datanode 将重新进

行块的分配；当数据块可以正常写入数据时，Datanode 通知客户端服务器文件写入成功，完成写操作，同时 Datanode 通知 Namenode 更新相应的元数据信息。相应的上传过程即小文件的写操作流程如图 2 所示。

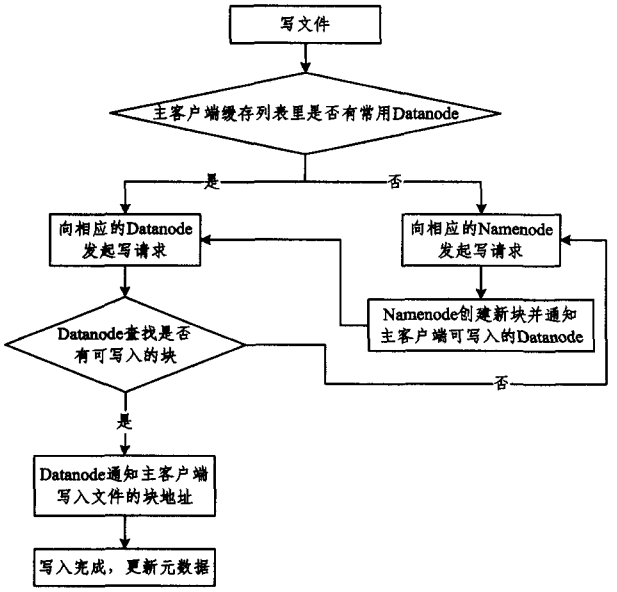

图 2　小文件的写操作流程

合并时需要指明小文件名和小文件相关的数据，如每个小文件的长度，小文件在大文件中的偏移量，小文件到大文件的映射关系。同时，为每个大文件创建一个局部索引文件[10]，用来记录属于该大文件的所有小文件的偏移量和长度，局部索引文件位于该大文件的每一个块的起始位置，并且只为这个大文件服务；并在客户端服务器的主存中保存相应的局部索引文件，在无修改的情况下读取小文件时可以直接查找小文件在大文件中的偏移及长度，无需与 Datanode 交互即可获得长度及偏移，可以进一步地减少 I/O 交互，提高访问效率。

3.2　预取与缓存

文件合并减少了元数据信息的数量，缓解了 Namenode 端的瓶颈问题，但没有提高小文件的读取效率。为了提高小文件的读取效率，提出了元数据信息缓存和关联小文件预取机制。1)元数据信息缓存：客户端服务器首次从 HDFS 上读取小文件时，首先向 Namenode 发起请求以获得大文件相应的元数据信息，根据元数据信息中的 Datanode 信息，与相应的 Datanode 进行交互，然后客户端将该元数据信息存入缓存中，如果该大文件中的其它小文件被请求时，则能够直接从缓存中读取相应的元数据信息，从而减少与 Namenode 的交互。2)关联小文件的预取：因为同一个大文件中的小文件都属于同一课程的关联小文件，当被请求的小文件返回到客户端时，会激发相应的预取机制，即将相关小文件预存至客户端缓存，当有相关小文件请求时，客户端只需从缓存中读取小文件，不用再与 Datanode 进行交互。当客户端服务器的缓存消耗 80%时，删除缓存中最长时间未访问的元数据信息或关联小文件。以上两种机制减少了 I/O 的访问次数，因而可以提高小文件的读取效率。

3.3　文件的读取操作

客户端请求读取小文件时，首先发送请求到客户端服务器，客户端服务器收到请求后，查找其缓存中的局部索引文件，找到小文件所在的大文件及小文件在大文件中的偏移及小文件的长度。然后，客户端服务器在其缓存列表里查找是否有相应的小文件缓存信息或元数据信息，如果有，则直接读取小文件或根据元数据信息查看相应的 Datanode 数据块信息，并到相应的块读取小文件，同时激活小文件预取机制；如果没有，则向 Namenode 请求获得与大文件相关的元数据信息，这个元数据信息包含了大文件所在的 Datanode 信息及数据块信息。得到元数据信息后，客户端服务器与相应的 Datanode交互，找到相应的块，并根据局部索引文件中小文件在大文件中的偏移量和小文件的长度读取小文件，返回到客户端，完成小文件的读取操作，同时激活关联小文件的预取策略。文件读取操作流程如图 3 所示。

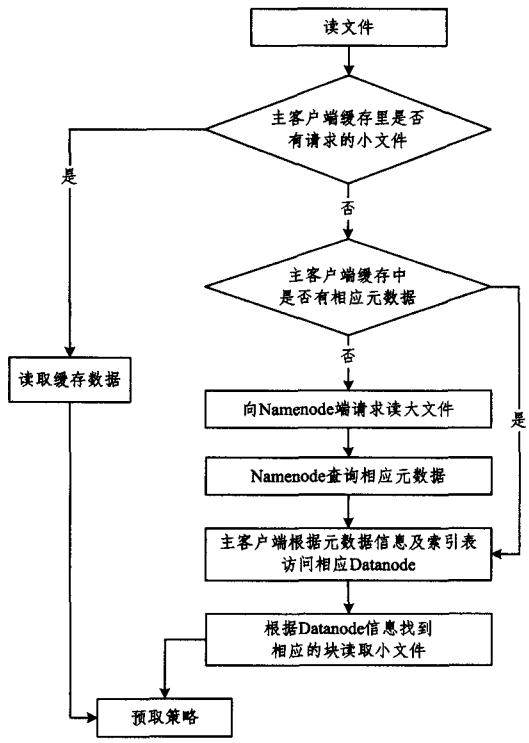

图 3　文件读取操作流程

3.4　碎片整理

所谓碎片整理指的是，对 Datanode 数据块中因某些小文件被删除或者其他原因造成的较小空间的空白存储区域的利用[11]。为了利用这些空闲存储空间，本文提出了碎片整理机制。该机制建立一个二叉平衡树数据结构的碎片索引，通过对这个索引的查找来找到碎片在数据块中的位置。碎片索引包括各碎片所在数据块中的偏移量和碎片的长度，按碎片长度将其放入相应的索引项中，通过对碎片索引中索引项的查找来定位碎片。另外，还可进行插入操作将相应大小的小文件存入数据块中，或进行删除操作，删除不需要的小文件，这些操作与二叉平衡树的查找、插入和删除操作相同。

当需要写入某类小文件时，找到小文件所属课程的大文件，通过查找大文件的碎片索引，查看该块中是否有适合的碎片可以存入小文件，若有大小刚好合适的碎片，则将小文件写入；若碎片大小大于小文件大小，则将该碎片分为两个部分，前一部分分配给待写入小文件，后一部分作为新的碎片，重新在碎片索引中生成相应的索引项，同时删除原碎片的索引项。

如果没有碎片可以写入小文件，则不对碎片索引进行任何修改，直接在数据块末尾的空白区分配相应大小的空间给小文件。

当删除小文件时，首先使用待删除小文件的文件名查找局部索引文件，判断是否存在该文件，若不存在，则删除失败；若存在，则在碎片索引中插入一条新的碎片索引项。判断新的碎片索引项的相邻存储空间是否同样是碎片数据，如果相邻空间中有空白索引，那么合并多个数据碎片成一个大的数据碎片，并更新碎片索引，删除原来的碎片索引项。但是，如果数据碎片的相邻碎片是由于数据块的分界造成时，则不需要进行数据碎片的合并。

4 实验结果和分析

本文设计了一个实验方案，用于测试在优化后的 HDFS 中小文件读写效率的变化以及所用内存开销的变化。本文在实验过程中以 HDFS 中的 Namenode 内存占用、文件的数量和大小、文件读取时间等数据作为实验参考指标，最后将实验结果与原 HDFS 中的实验结果进行对比分析。实验环境：操作系统(ubuntu 12.04 64bit)、处理器(intel 双核 2.4GHz)、内存(4GB)、硬盘(250G 5400r/min)、网络(1.0 Gbps 以太网)、JDK 版本(1.7.45)、Hadoop 版本(0.2 0.3)、Namenode 节点数目(1)、Datanode 节点数目(3)。

4.1 内存占用实验结果与分析

为了分析基于 HDFS 的分布式文件存储系统中 Namenode 的内存占用情况，本文设计了以下实验方法。

选取 50000 个大小为 1～100kB 的小文件作为实验数据集，将这些小文件依次以不同的数量分别存储在原 HDFS 系统中和经过优化的 HDFS 系统中。每次启动 Namenode 后，分析随机读取文件时 Namenode 的内存占用情况。

根据 Namenode 所占用系统内存的变化并取其平均值得到的优化后 HDFS 与原 HDFS 的 Namenode 占用内存大小的对比情况如图 4 所示。

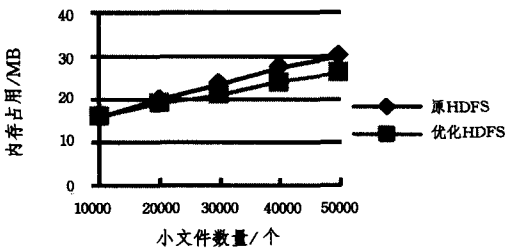

图 4 Namenode 占用内存对比

从 Namenode 的内存占用图中可以看出，当小文件数量不超过 10000 时，本文设计的优化方案效果并不明显；但是随着小文件数量的不断增加，尤其是文件数量超过 30000 后，优化后的 HDFS 中 Namenode 所占用的系统内存比原生 HDFS 有所节省，尤其是文件数量较大时，效果更为明显。优化后的 HDFS 能够节省内存的主要原因在于，本文提出的优化方案采用了文件合并策略，先将小文件合并成大文件再进行存储，使得元数据的数量大幅度减少，从而有效减少了 Namenode 的内存占用。

4.2 文件的数量和大小分布

为了观察小文件的大小及数量的整体分布情况，将这 50000 个文件以 5 种不同的数量规模分别存储在原 HDFS 系统和优化后的 HDFS 系统中。图 5 给出了关于小文件大小及数量的分布情况。

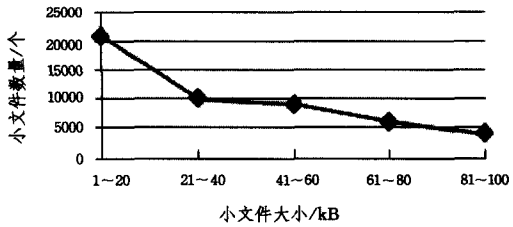

图 5 小文件大小及数量分布

从以上小文件的数量及大小分布图可以看出，大小在 1～20kB的文件占了 50%，大小在 21～40kB 的文件占 25%，可以体现这类大小的文件在教育资源中大量存在，不适合直接存在 HDFS 上，将其合并成大文件后进行存储可提高效率。

4.3 读写性能结果及分析

根据本文提出的优化方案，首先对小文件建立索引，根据相应的合并规则对这些小文件进行合并存储。其中，小文件分类及索引处理时间记为 T_{sort}，存入 HDFS 的时间记为 T_{store}，小文件总数量为 Num。则得到小文件的平均存储时间 T 为：

$$T=(T_{sort}+T_{store})/Num$$

对于原 HDFS 系统，由于小文件在存入 HDFS 之前不存在合并及建立索引之类的工作，因此上传到 HDFS 的时间 T' 即为小文件存储到 HDFS 中的时间。

$$T'=T'_{store}$$

因此，同样选取 50000 个大小为 1～100kB 的小文件作为实验数据集。将这些小文件以不同的数量规模存入 HDFS 中，对相应的存储过程重复 3 次，最终以这 3 次的平均时间作为小文件存储时间；然后再将这些文件全部从 HDFS 中读取出来，同样重复 3 次，将这 3 次操作的平均时间作为小文件的读取时间。图 6 示出了小文件的平均读取时间对比。

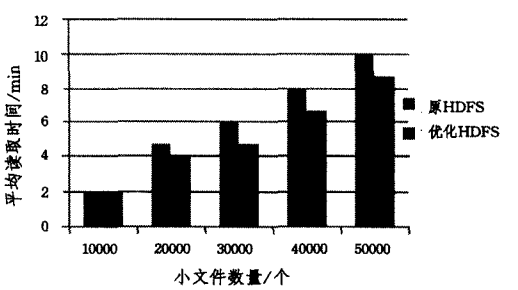

图 6 平均读取时间对比

通过观察图 6 可以发现，在小文件读取方面，优化后的 HDFS 的小文件读取性能略有提升，其原因归结于 Namenode 在文件读取时查询速度的提升。但是对于小文件的存储，优化后的 HDFS 的写入时间相对于原 HDFS 有明显缩短，原因有两个方面：首先，虽然小文件在存入 HDFS 之前需要时间建立索引与小文件合并，但随着小文件索引的合理优化，这些操作额外消耗的时间相对于存储时间微乎其微；其次，本文提出的优化方案采用的文件合并策略大幅度减少了 Namenode 中元数据的数量，因而小文件在写入 HDFS 时 Namenode 检

索 HDFS 空闲块的时间随着元数据的减少而减少。

结束语 HDFS 是为存储大文件而设计，当存储大量小文件时会引起 I/O 性能瓶颈，同时，Namenode 主存里的元数据也会迅速增长，造成瓶颈。本文提出的合并及缓存策略将大量关联小文件合并成大文件后存储于 HDFS 上，有效缓解了 Namenode 主存的瓶颈问题；通过关联小文件及元数据的预取方案，有效地提高了 I/O 性能，解决了 HDFS 上存储小文件存在的问题。对 50000 个小文件进行实验，结果表明通过小文件合并及相关缓存预取策略使内存占用相对原始 HDFS 系统减少了 19%，读取效率相对原始 HDFS 提高了 20%。接下来将进一步改善解决方案中的合并策略，进一步研究小文件之间的联系从而尽可能地将属于同一类课程的小文件合并到一个大文件中，将关联关系最强的小文件放在相邻存储空间，以更好地实现关联小文件的预取策略，提高小文件预取准确率，进而更好地改善小文件在 HDFS 上的读取效率。

参 考 文 献

[1] kkdelta. 告诉你 Hadoop 是什么[EB/OL]. [2014-06-17]. http://www.thebigdata.cn/Hadoop/10722.html

[2] 周敏奇，王晓玲，金澈清，等. Hadoop 权威指南(第 2 版)[M]. 北京：清华大学出版社，2011：8-20

[3] White T. The small files problem [EB/OL]. [2009-2-2]. http://www.cloudera.com/blog/2009/02/the-small-files-problem

[4] Dong Bo, Qiu Jie, Zheng Qing-hua, et al. A novel approach to improving the efficiency of storing and accessing small files on Hadoop: a case study by powerpoint files [C]// IEEE International Conference on Services Computing. Miami, Florida, Piscataway: IEEE, 2010: 65-72

[5] 李宽. 基于 HDFS 的分布式 Namenode 节点模型的研究 [D]. 广州：华南理工大学，2011
Li Kuan. Research of the Model of Distributed Namodes in HDFS[D]. Guangzhou: South China University of Technology, 2011

[6] 赵晓永，杨扬，孙莉莉，等. 基于 Hadoop 的海量 MP3 文件存储架构[J]. 计算机应用技术，2012，32(6)：1724-1726
Zhao Xiao-yong, Yang Yang, Sun Li-li, et al. Hadoop-based storage architecture for mass MP3 files[J]. Journal of Computer Applications, 2012, 32(6): 1742-1726

[7] Fu Song-ling, Huang Chen-lin, He Li-gang, et al. iFlatLFS: Performance Optimization for Accessing Massive Small Files[C]// 20th International Conference on High Performance Computing. Bangalor, Piscataway: IEEE, 2013: 10-19

[8] Li Jia, Lin Kun-hui, Wang Jing-jin. Design of the Mass Multimedia Files Storage Architecture Based on Hadoop[C]// the 8th International Conference on Computer Science & Education. Colomlo, Piscataway: IEEE, 2013: 801-804

[9] 王涛，姚世红，徐正全，等. 云存储中面向访问任务的小文件合并与预取策略[J]. 武汉大学学报(信息科学版)，2013，38(12)：1504-1508
Wang Tao, Yao Shi-hong, Yu Zheng-quan, et al. A Small File Merging and Prefetching Strategy Based on Access Task in Cloud Storage[J]. Geomatics and Information Science of Wuhan University, 2013, 38(12): 1504-1508

[10] Chandrasekar S, Dakshinamurthy R, Seshakumar P G, et al. A Novel Indexing Scheme for Efficient Handling of Small Files in Hadoop Distributed File System[C]// 2013 International Conference on Computer Communication and Informatics (ICCCI). Coimbatore, Piscataway: IEEE, 2013: 1-8

[11] 郑庆华，董博，刘均，等. 一种基于 Hadoop 的海量可归类小文件关联存储方法：中国，102332029A[P]. 2012-01-25

(上接第 49 页)

[5] Platform specific meta-model of deployment environment of Google App Engine[OL]. http://download.csdn.net/download/u012261044/8386821

[6] Bunch C, Drawert B, Chohan N, et al. Language and Runtime Support for Automatic Configuration and Deployment of Scientific Computing Software over Cloud Fabrics[J]. Journal of Grid Computing, 2012, 10(1): 23-46

[7] Brandtzæg E, Mohagheghi P, Mosser S. Towards a Domain-Specific Language to Deploy Applications in the Clouds[C]// CLOUD COMPUTING 2012: The Third International Conference on Cloud Computing, GRIDs, and Virtualization. 2012

[8] Wettinger J, Andrikopoulos V, Strauch S, et al. Enabling Dynamic Deployment of Cloud Applications Using a Modular and Extensible PaaS Environment[C]// Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing. 2013: 478-485

[9] D'Andria F, Bocconi S, Cruz J, et al. Cloud4SOA: Multi-cloud Application Management Across PaaS Offerings[C]// Proceedings of the 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'12). 2012: 407-414

[10] Quinton C, Haderer N, Rouvoy R, et al. Towards multi-cloud configurations using feature models and ontologies[C]// Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds (MultiCloud'13). 2013: 21-26

[11] Ferry N, Rossini A, Chauvel F, et al. Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems[C]// Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD'13). 2013: 887-894

[12] Nguyen B M, Tran V, Hluchy L. Development and deployment of cloud services via abstraction layer[C]// 2013 International Conference on Computing, Management and Telecommunications (ComManTel). 2013

[13] Sellami M, Sami Yangui, Mohamed M, et al. PaaS-Independent Provisioning and Management of Applications in the Cloud[C]// Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD'13). 2013: 693-700

[14] Ma Zhi-yi, He Xiao. Building Modeling Tools Based on Metamodeling and Product Line Technologies[J]. Chinese Journal of Electronics, 2014, 23(2): 219-226