Hadoop 环境下三维模型的存储及形状分布特征提取

李海生 赖 龙 蔡 强 毛典辉 陈 谊

北京工商大学计算机与信息工程学院，北京 100048

(lihsh@th.btbu.edu.cn)

Research on 3D Model Storage and Shape Distribution in Hadoop Environment

LI Haisheng, LAI Long, CAI Qiang, MAO Dianhui, CHEN Yi

(School of Computer and Information Engineering, Beijing Technology and Business University, Beijing 100048)

Abstract With the explosion of 3D models, how to organize, manage and browse large-scale model files effectively has become a critical problem. The 3D model files are classified and merged firstly based on the concept of model file names' similarity, and then deposited into Hadoop distributed file system (HDFS). 3D model shape distribution is realized according to MapReduce parallel programming method in Hadoop environment by introducing the topology consistency factor and 3D model integrity function. In addition, the performance of MapReduce process is optimized according to the characteristic of distributed computing. Taking NTU3d data set as testing, the validity of proposed algorithm is verified in Hadoop environment.

Key words 3D model; Distribute File System; MapReduce; Shape Distribution

摘要 随着三维模型数量爆炸式的增长，如何有效地存储和管理海量的三维模型文件并对其进行高效的处理，是三维模型检索领域亟待解决的问题。本文首先基于模型文件名的概念相似度对模型文件分类合并，存入 Hadoop 集群的分布式文件系统(HDFS)。通过引入拓扑结构一致性因子，设计了三维模型完整性函数，实现了 Hadoop 环境下对三维模型形状分布的 MapReduce 处理，并根据分布式计算的特点进行了性能优化。以中国台湾大学的三维模型数据库作为测试集在 Hadoop 集群上进行实验，验证了本文算法的有效性。

关键词 三维模型；分布式文件系统；MapReduce；形状分布算法

中图法分类号 TP391

1 引言

近年来，三维模型扫描设备以及三维重建技术在工业设计、文物保护、游戏娱乐、影视动画、医学仿真等领域得到广泛应用，此外 Pro/E、UG、Maya、3dMax 等三维造型软件也得到业界的普遍使用，导致三维模型的种类和数据量与日俱增。如何有效地存储和处理海量的模型文件，为三维模型重用以及后续研究工作做好准备，是当前亟待解决的问题。到目前为止，已经有很多研究机构提出过针对三维模型进行存储和处理的三维模型检索系统。其中具有较大影响力的有美国普林斯顿大学提供的三维形状搜索引擎（3D Model Search Engine）[1]、普渡大学（Purdue University）的三维形状基准（Purdue 3D Shape Benchmark）[2]和台湾大学提供的三维形状检索系统（3D Model Retrieval System）[3]等。但是目前针对三

收稿日期：2014-11-12
基金项目：国家“九七三”重点基础研究发展计划项目(2012CB821206);国家自然科学基金(61320106006);虚拟现实技术与系统国家重点实验室开放基金资助(BUAA-VR-14KF-04);北京市属高等学校教师队伍建设--青年英才计划资助项目(YETP1452)

维模型的研究集中于基于传统存储和处理技术来构建三维模型检索系统，在面对海量的三维模型文件时，无法保证检索的时效性。因此将三维模型检索与分布式存储、处理模型文件相结合，解决海量三维模型的存储和处理，具有重要的现实意义和应用价值。

为了应对海量数据存储与数据挖掘的挑战，谷歌在 2003 年首先提出了谷歌分布式文件系统 GFS 的技术思想[4]，随后 2004 年谷歌进一步提出 MapReduce 并行编程框架[5]。基于 GFS 和 MapReduce 编程思想，开源社区 Apache 发起了 Hadoop 项目，实现了分布式文件系统 HDFS 和并行编程框架 MapRedcue。Hadoop 的分布式文件系统 HDFS 运行于高可靠的商用硬件集群上，以流式数据访问模型来存储超大文件[6]，以易于编程、良好的扩展接口、高容错性为设计目标[7]。使用基于 HDFS 的三维存储策略，可以更加有效的对现有的三维模型进行管理，提高用户在进行检索时的实时交互性。在处理大规模数据方面，MapReduce 是当前最优秀的的并行编程模型之一[8]。该项目已在移动互联网、电子商务、社交网络等领域得到广泛应用。由于三维模型具有数据量大、增长速度快、结构种类多、隐含知识丰富等特点，使用 Hadoop 平台可以更高效的应对大规模三维模型的处理。当前 MapReduce 在图形领域的应用研究处于探索性阶段。Brian Summa 基于 MapReduce 框架实现了 z-buffer 绘制以及等值面抽取等基础的图形学算法，讨论了基于 MapReduce 的图形学算法并行化。阿里云计算公司与浙江大学 CAD&CG 国家重点实验室推出了渲染云计算[9]，基于“飞天”大规模分布式计算系统提供弹性自助式渲染农场服务。

本文通过分析三维模型文件大小差异对集群性能影响，结合三维模型文件结构的特点，提出了基于 HDFS 的三维存储与管理策略，设计实现了使用 MapReduce 对具有严格结构要求的三维模型文件处理的方法。首先利用三维模型文件操作的数据局部性原理，对模型初步分类，将相似度大的模型文件合并后进行数据块封装，缓解了三维模型文件大小差异对集群带来的性能瓶颈；其次，分析了使用 MapReduce 处理三维模型时的顶点位置依赖性问题，并使用 MapRedcue 对三维模型进行标准化处理，改进了传统的单机模式的三维模型形状分布特征提取算法，最后提出了 MapReduce 三维模型处理程序性能调优策略，提高了作业执行效率。

2 面向特征提取的三维模型的 HDFS 存储

Robert Osada 等提出的形状分布算法[10]，也叫 Osada 算法，对任意的三维多边模型计算形状特征，是三维模型检索领域最为经典的算法之一。本文分析 Osada 算法，将其与 MapReduce 编程模式有机结合，设计出适合在 Hadoop 平台上进行批量特征提取的三维模型检索算法，以此来优化三维模型检索系统。

Osada 形状分布算法关键思想是从三维模型的全局几何性质的函数上采样得到的形状分布直方图作为该三维模型的特征向量。然后再根据特征向量进行三维模型间的相似度计算，其过程如图 1 所示。

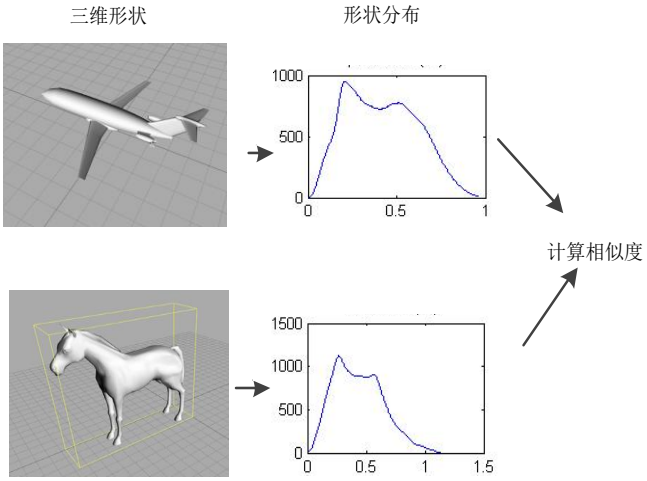

图 1 形状分布算法思路

Osada 特征提取算法单独提取一个模型特征向量的基本思想是：首先规定好要采样点的总数、统计直方图的区间数以及提取特征向量的维数；然后对三维模型预处理，将三维模型的重心移动到原点，整个模型缩放到单位圆内；接着计算每个三角面片的面积并排序处理，计算三维模型的表面积，采用基于面积加权的随机采样策略对模型进行采样。根据三维模型的表面积生成随机数 R（R 为小于表面积的正数），将排好序的三角面片面积从小到大累加，设前 i-1 个三角面片面积累加值小于 R 且前 i 个三角面片面积累加值大于 R，则在三角面片 i 上随机取一点 P. 点 P 的选取如式 1 所示。

$$P=(1-\sqrt{r_1})\,A+\sqrt{r_1}(1-r_2)\,B+\sqrt{r_1}r_2C \quad (式1)$$

其中 A、B、C 分别为三角面片三个顶点，r1、r2 为[0,1]间的随机数；然后计算模型上任两采样点之间的欧式距离，统计该距离的分布频率。由于预处理过程将模型缩放到单位圆内，统计分布区间为[0,2]；之后对统计的分布频率进行函数拟合，最终生成维数为 dimNumber 的向量作为模型的特征向量。具体的算法流程如图 2 所示。

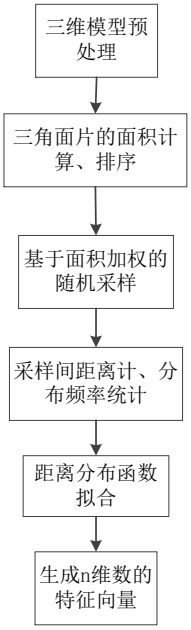

图 2 单个模型形状分布特征提取算法流程

目前的三维模型检索系统的文件存储都是基于传统的单点文件系统，不能满足日益增长的三维模型数量对存储容量的需要。同时大量三维模型的处理带来庞大的计算量，传统的文件系统、单点架构的存储能力以及计算能力无法满足要求。而使用 Hadoop 存储、处理三维模型时存在一系列问题。一方面，由于三维模型文件大小差别很大，从 KB 级直到上百 MB，而且存在众多小模型文件，给 HDFS 存储三维模型文件带来挑战。另一方面，三维模型文件结构具有特定的逻辑，三角面片的数据依赖于顶点的位置信息，MapReduce 过程中顶点位置的变动会破坏模型文件，这是 MapReduce 处理三维模型需要解决的问题。

为了解决三维模型文件大小差异的问题，本文首先将三维模型初步分类，然后对众多三维模型文件进行合并，构建索引后分别将模型文件和索引文件存入 HDFS。如图 3 所示。

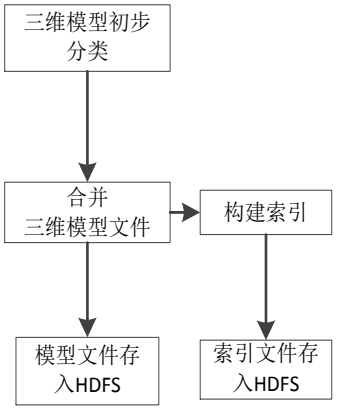

图 3 三维模型 HDFS 存储思想

2.1 三维模型文件大小差异问题分析

通过调研发现，普林斯顿大学、台湾大学等的模型库中文件大小差别很大，既有 20KB 的超小文件，又有 80MB 的较大文件，但是其中有 95%的模型文件都小于 10MB。关于小文件的评判标准，一些学者进行了相应的研究。Liu 等认为小于 16MB 的文件是小文件[11]。Bo Dong 等通过实验量化得出在其实验环境下，小于 4.35MB 的文件是小文件[12]。小文件对 Hadoop 集群带来的负面影响已经引起学者、开发人员注意，并进行了相应的研究工作[12]。在 Hadoop 集群中，NameNode 将文件的元数据存储于内存中，每个文件、目录和数据块的存储信息大约占 150 字节。当文件数量达到百万千万，且每个文件占一个数据块，内存占用量会给 NameNode 节点带来内存瓶颈。而在 MapReduce 处理大量的小文件时，会为每个存储了一个小文件的数据块（小于 slot）启动一个 map。过多的 map 会影响任务的提交速度，延长任务的运行时间，浪费集群系统资源。模型文件大小差异明显，既会导致 map 数目不确定，出现数据倾斜，也会延长作业的运行时间，并导致模型存储的不一致性。

因此如何处理小文件，是在 HDFS 中存储三维模型的关键技术[13,14]。为了达到存储负载均衡的同时便于三维模型检索操作，本文先根据模型文件的语义信息相似度对模型文件进行粗分类，然后对同类模型文件进行合并。

2.2 三维模型概念相似度分类

三维模型检索遵从数据的局部性原理，这里的局部性指的是模型间的相似度、文件大小等，例如检索三维模型时，用户更关注与之相似或属于同一类别的模型。因此相同类别的模型将被归并到一处后存入 HDFS 中，以便于检索。普林斯顿大学三维模型检索小组提出了通用三维模型的基本分类信息[15]，本文在此基础上，为所有类别设置一个共同的根节点 object，构建一棵分类树，如图 4 所示。该分类树共有 92 个叶子类，总类别有 132 个，本文使用该分类标准对收集到的三维模型进行初步分类。

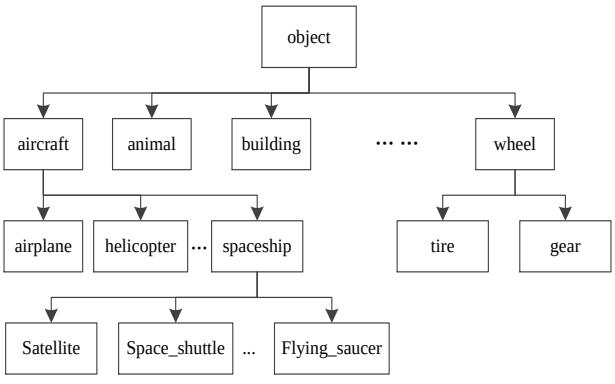

图 4 三维模型初步分类体系

从网络上获取三维模型，通常取最能代表该模型的语义信息作为模型文件的名字，所以本文利用语义相似度模型进行分类。目前，国内外学者已经对概念相似度计算进行了广泛的探索和研究，提出了很多计算相似度的方法[16]，可分为基于路径的方法和基于信息内容(Information Content, 简称 IC)的方法。本文采用普林斯顿大学开发的 WordNet 词典提供的接口计算概念相似度[17]。三维模型通常使用名词标识其语义信息， WordNet 利用名词间的“is-a”关系构成上位词路径。

但是利用 WordNet 的词库计算相似度十分耗时，本文充分利用“减枝”策略对基于概念相似度的模型初步分类算法进行加速。引入“爬山算法”的思想，计算排名前 3 的类别的子类别与该模型的概念相似度，一定程度上避免了相似度最高而没有排到第一位的情况，确保算法的稳定性。基本流程是广度优先遍历分类树，用队列辅助对分类树遍历，从队列中取出一个类别，计算模型名与该类别间的相似度，放入一个容量为 3 的数据结构 SizedSet 中，如果放入成功，将其子类别每层取相似度排前三的类别，放入队列中， 如果下一层中的相似度小于上一层得到的相似度，则对该结点“减枝”。最后，将模型归为与其相似度最大的类别。具体流程如算法 1 所示。

算法 1. 模型概念相似度分类算法

输入 ： 分类树 CategoryTree、模型文件 model
输出 ： 模型归属类别 category

```
classified(CategoryTree, model)
{
    initial queue;                                  //初始化空队列
    add(queue, CategoryTree. getChildArray());      //将分类树根节点的孩子节点加入到队列 queue
    initial sizedTopSet(elemt1, elemt2, elemt3);    //初始化一个容量为 3 的数据集合，用于保留与输入模型相似度最大的类别信息
    category = queue;                               //记录指针在队列中所指向的元素
    while(*category != NULL)                        //如果 queue 队列未遍历完，继续循环
    {
        simrlt = sim(category, model);              //使用 WordNet 方法计算当前类别与输入模型名称相似度
        if(simrlt>min(sizedTopSet))                 //如果得到的相似度大于 sizedTopSet 中三个元素的最小值
        {
            delete min(sizedTopSet);                //删除 sizedTopSet 中值最小的元素
            add(sizedTopSet, model);                //将当前模型的信息记入 sizedTopSet 中
        }
        add(queue, category. getChildArray());      //将当前类别的子节点加入 queue 队列
        Category++;
    }
    return max(sizedTopSet);                        //返回 sizedTopSet 中值最大的节点
}
```

2.3 分类模型合并存储策略

三维模型进行概念相似度分类之后，将相同类别的模型文件合并以减少文件数目，减轻中心节点因小文件过多带来的内存以及 CPU 负担。针对同一类别的模型文件大小差异，本文采用小文件优先的原则归并模型文件，使得最终得到的序列文件 SequenceFile 的数目最小。一个模型类别可能会有多个 SequenceFile，若文件 m 符合（式 2）的不等式组则将 m 归入当前的 SequenceFile，其中 f_i 表示文件 i 的大小。

$$\begin{cases} \sum_{i=1}^{m-1} f_i < 64\text{MB} \\ f_m + \sum_{i=1}^{m-1} f_i > 64\text{MB} \end{cases} \quad \text{（式 2）}$$

本文为合并后的模型文件设计了二级索引。第一级索引对应模型的类别，通过类别索引到对应的 SequenceFile。每个 SequenceFile 对应一个二级索引文件，用来对合并的模型文件进行索引。合并模型文件并构建索引的思路如图 5 所示。将分类的模型文件按小文件优先的原则生成 SequenceFile，输出索引。在一级索引中按类别索引到模型所在的.seq 文件，在二级索引中通过文件名找到在.seq 文件的位置和大小。

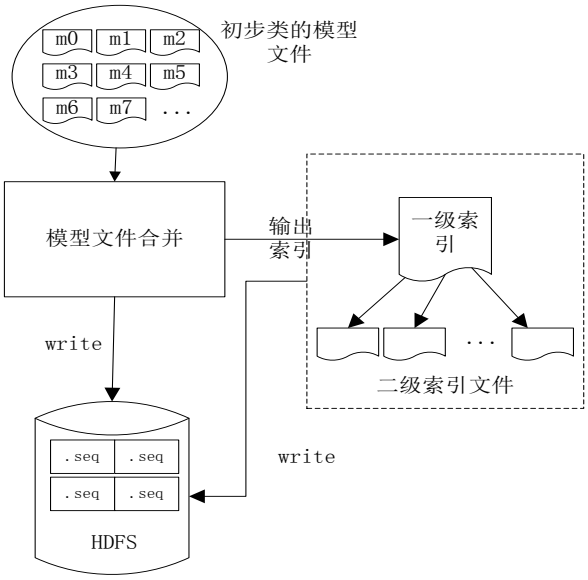

图 5 模型文件合并生成索引的流程图

3 MapReduce 环境下处理三维模型

3.1 三维模型文件结构的特殊性

三维模型文件的内部顶点坐标组成一个序列，三角面片以及纹理、颜色、顶点法向量等属性之间的关系都是用顶点序列的索引位置确定的。图 6 为一个 obj 格式的三维模型文件(tree.obj)截图。

```
# This file uses centimeters as units for non-parametric coordinates.

v 0.061043 0.025284 0.034490
v 0.011829 0.022302 0.083267
......
v -0.348501 0.914088 -0.137929
vt 0.131375 0.762327
vt 0.437504 0.963420
......
vt 0.564338 0.688425
vn 0.912731 0.155721 0.377721
vn 0.912731 0.155721 0.377721
......
vn 0.224233 0.922361 0.314595
f 37/262/1 38/1103/2 46/1104/3
f 46/1104/3 38/1103/2 1/9/4
......
f 378/907/1634 407/967/1635 402/942/1636
```

图 6 obj 格式的 tree 模型的部分数据

在 obj、off 等格式的三维模型文件中，三角面片是用顶点位置索引表示的，使得三维模型文件顶点位置结构不可变动。要将三维模型的处理操作移植到 MapReduce 上，需要充分考虑 MapReduce 处理流程和三维模型文件的结构特征。

MapReduce 作业的执行流程可分为五个阶段：FileSplit 阶段、Map 阶段、Partition 阶段、Combine 阶段、Reduce 阶段。MapReduce 通过对流式文件无差别的批处理过程，将文件记录无序化处理，通过集群批处理提高处理大数据的能力。MapReduce 程序对三维模型进行处理时，在 map 阶段结构固定的模型文件被分成众多无序记录，这样就丧失了模型文件保持的顶点间的相互位置信息，顶点索引出现混乱，模型的完整性遭到破坏。因此需要考虑顶点位置结构的不可变动性，编写合理的 MapReduce 程序对模型进行处理。

3.2 批量处理的形状分布算法

根据为单个模型特征提取的算法思想，本文针对 MapReduce 无序化处理文件记录的形式，提出了分散化的可批量对模型库模型进行基于形状分布的特征提取算法。在 Job 层面的基本算法流程如图 7 所示，Job1 中进行模型文件标准化，Job2 中对三维模型随机采样，Job3 中进行采样点间距离计算、分布统计、函数拟合最后生成特征向量。

MapReduce 将三维模型文件的每行打散，为保证模型拓扑结构的一致性和完整性，本文提出了模型结构拓扑一致性因子 S 和模型完整性函数 D。其中 S 为一个 $(id, flag, dist)$ 组成的三元组，id 是模型的唯一标识，flag 是面片或者顶点的标记，dist 是顶点据起始位置的长度。模型性完整性函数 $D = f(S)$，保证被打散的属于同一个模型的数据在 Reduce 阶段由同一 Reducer 进行处理。

图 7 MapReduce 重写形状分布特征提取算法 Job 层面流程图

3.2.1 Job1 模型库模型文件标准化的算法设计

由于三维模型的尺度、方位以及旋转角度等的不一致可能会对提取得到的特征以及相似性度量造成影响，因此在提取特征之前，需要对三维模型进行标准化。常见的标准化方法主要对模型进行平移、旋转和缩放处理，将所有待比较的模型变换到一个规范坐标系内，对模型的大小进行统一。

Hadoop 环境下的三维模型标准化算法流程如图 8 所示。拓扑结构一致性因子采用自定义类型 TextTextLongTriples 进行描述。在 Mapper 端，对每个模型文件按行读取，顶点信息 flag 为 0，面片信息 flag 为 1。将模型文件名、标记信息和 Mapper 的输入 key（即该条记录所在行数），作为 Mapper 输出的 key 值。同时将顶点、面片信息形成字符串作为 Mapper 端输出的 value。

模型完整性函数 KeyPartitioner，将具有相同文件名的所有记录分组到一个 Reducer 中。由于自定义 TextTextLongTriples，同一个 Reducer 处理的记录按标记信息排序，顶点在前、面片在后，顶点会按 Mapper 输入的 key 值排序，保留了三维模型顶点间固有的顺序结构，解决了三维模型 MapReduce 处理的文件结构不可变动的障碍。

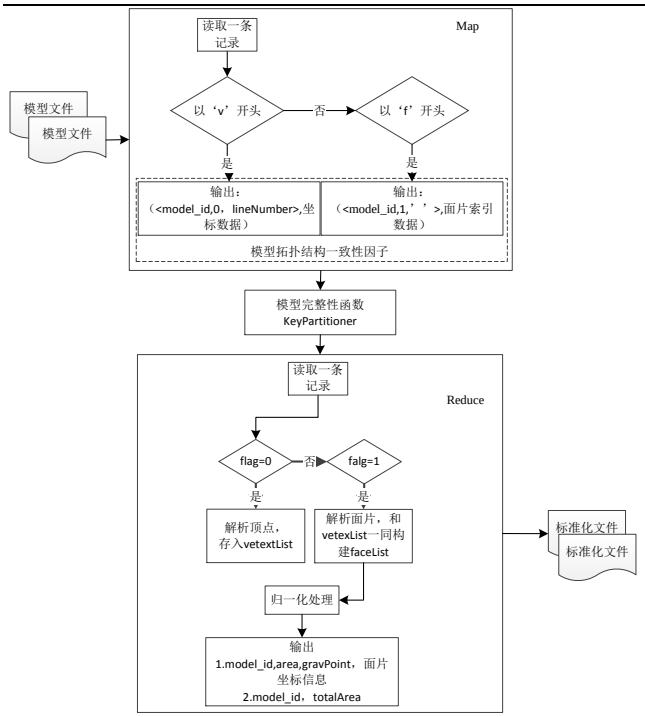

图 8 模型库模型文件标准化流程图

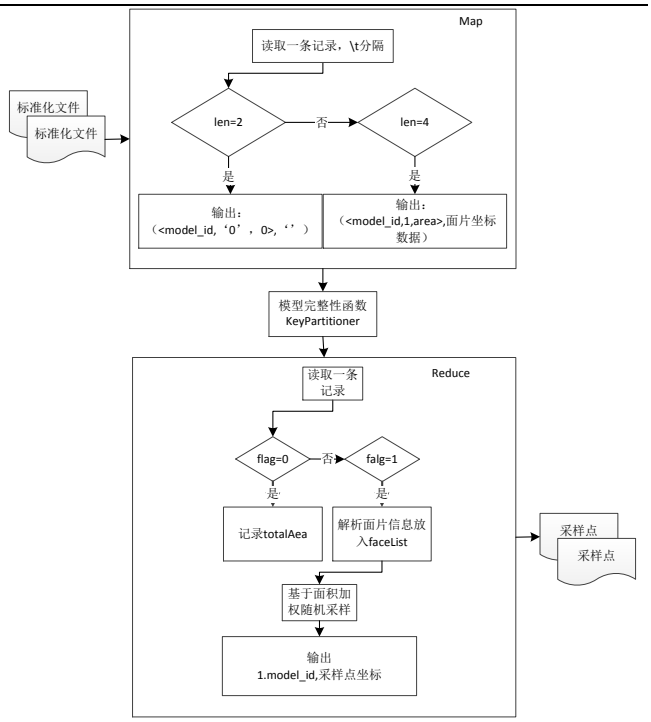

图 9 三维模型采样算法流程图

在 Reducer 端，先解析顶点，放入 vetexList 中，然后解析面片信息放入到 faceList。根据模型上各顶点坐标求出三维模型的重心，在标准化中将模型的重心平移到坐标原点，并将模型缩小到单位圆内。最后在 Reduce 端输出两种格式的数据，作为 Job2 的输入，输出格式：

第一种：model_id+’\t’+area+’\t’+gravPoint +’\t’+x1 y1 z1,x2 y2 z2,x3 y3 z3

第二种：model_id+’\t’+totalArea

3.2.2 Job2 三维模型随机采样

MapReduce 环境下三维模型随机采样流程图如图 9 所示。

Mapper 端，由于输入的数据有两种格式，对于第一种格式的数据，将 model_id、’1’、面片面积组装出一个自定义的 TextTextDoubleTriples 变量，同面片坐标信息一起作为一个 map 函数的输出；对于第二种格式的数据，用 model_id、’0’、0 组装成 TextTextDoubleTriples 型变量作为 key，value 设置为空，作为 map 的输出。

同 Job1 一样，模型完整性函数 KeyPartitioner 将具有相同文件名的所有记录分组到一个 Reducer 中，同一个模型的记录在 Reduce 之前会按照面片的大小进行归并排序。

Reducer 端，通过标记信息，得到模型的总面积 totalArea，以及模型各个面片的信息，然后解析面片放入 faceList。然后进行基于面积加权的随机采样。Reduce 的输出格式：

model_id+’\t’+x y z，作为 Job3 的输入。

3.2.3 Job3 提取特征向量

MapReduce 环境下提取特征向量流程图如 10 所示。

Mapper 端，扫描 Job2 的输出，直接封装到 Text 类型中，即 model_id 作为 key，采样点作为 value.经过默认的 Partitioner，同一模型的采样点将由同一个 Reducer 进行处理。

在 Reducer 端，首先计算采样点两两之间的欧式距离，统计距离的分布频率；然后对统计的分布频率进行函数拟合；最后在拟合成的函数上抽取维数为 dimNumber 的向量作为模型的特征向量。

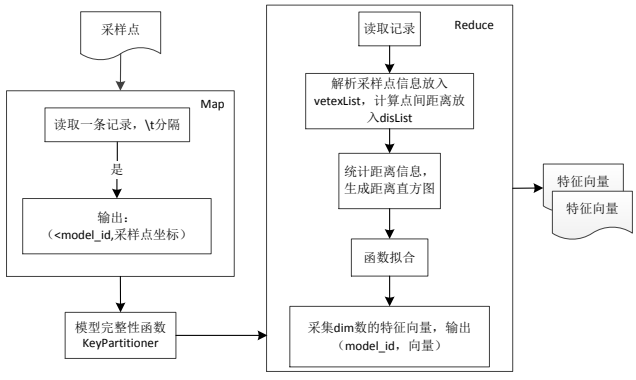

图 10 三维模型特征向量采集

4 实验结果与分析

4.1 实验环境

实验平台是由 4 个节点组成的 Hadoop 集群。一个 Namenode 节点和 3 个 Datanode 节点，其硬件与软件配置如 s 表 1 所示。

表 1 系统实验环境软硬件配置

	Namenode	Datanode
CPUs	16 Intel(R) Xeon(R) 2.40GHz	
内存(RAM)	24GB	12GB
硬盘大小	1TB	300GB
操作系统	Red Hat Enterprise Linux Server 6.1 (x86_64)	
JDK 版本	jdk-6u32-linux-x64	

Hadoop 集群中 4 个节点都位于同一个机架，共享一个千兆交换机，备份数设置为 3，使用默认数据块大小 64MB。

本文以中国台湾大学的 NTU3D 模型数据库[3]作为测试集，该数据库中每个模型有两个文件，一个是其 obj 格式的模型文件，另一个是 jpg 格式的模型图像。一共 21814 个文件，共 3.23GB，其中最大文件为 19,744KB，最小为 1KB。

4.2 文件数量对 Hadoop 集群影响

为了分析小文件对 Namenode 节点带来的影响，将多个测试集存入 HDFS，记录 Namenode 的 Java 内存使用量，如图 11 所示。本文使用 JConsole 记录了每存入一个测试集后执行垃圾回收操作的非堆内存和堆内存大小。从图中可以看出随着文件数量的增加，Namenode 节点的堆内存负荷呈线性增加，而非堆内存的负荷基本保持不变。

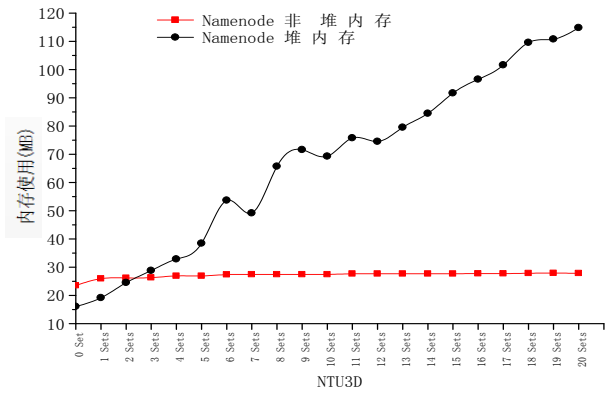

图 11 存储大量 NTU3D 模型库小文件对 Namenode 节点内存的负荷

以测试集中大小为 86MB 的模型 blade 为测试对象，通过使用 NLineInputFormat 作为 MapReduce 的 InputFormat，改变其默认的切分方式，测试 mapper 数对作业 Job 的影响。如图 12 所示，作业的运行时间会随着 Map 数的增长急剧延长。HDFS 最初是为流式访问大文件开发的，如果访问大量小文件，需要不断的从一个 Datanode 跳转到另一个 Datanode，增加了对 HDFS 的读取与写入操作，使得整个作业时间延长。每一个小文件要占用一个 slot，而 task 启动耗费时间，极端情况下整个 Job 大部分时间都耗费在启动和释放 task 上。即使 mapper 数的增加会使 Reduce 的 Copy 和 Sort 操作提前，但是节省下的时间远小于对 HDFS 频繁读写而延长的作业时间。

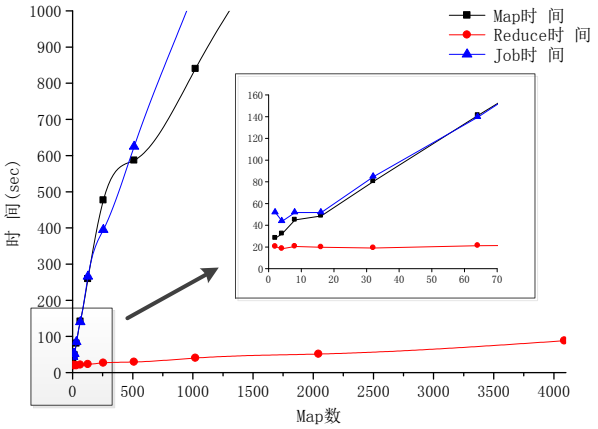

图 12 不同 Map 数对 Job 执行时间的影响

4.3 模型文件合并以及小文件的确定

为了测试模型文件合并对内存的影响，对 Namenode 节点内存的压力减小程度进行实验。图 13 为模型文件合并与不合并对 Namenode 节点内存的影响情况。可以看出，合并模型通过降低文件数量，减少了存储在 Namenode 节点内存的文件元数据信息，从而使 Namenode 节点的内存负荷减小，以便在 HDFS 上存储更多的模型文件。

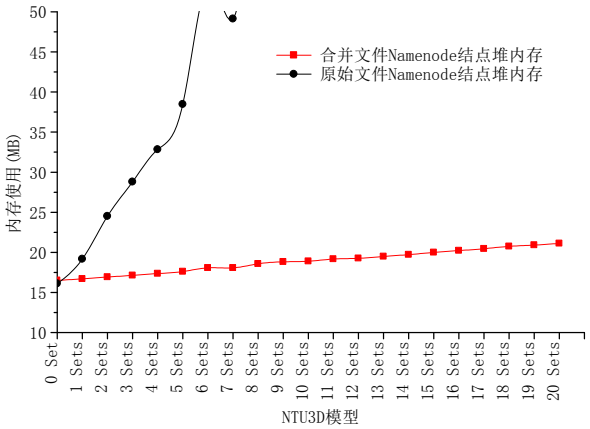

图 13 合并模型文件对中心节点内存占用的缓解

为了确定当前的实验环境所适合的小文件评判

标准以便于进行文件的合并，进行了下面一组实验。在测试集中选取了 15 个不同大小的三维模型文件组成子测试集。编写 MapReduce 程序对 15 个模型文件进行处理。通过统计处理各三维模型的 Job 计数器信息，得到 Job 的各个阶段不同模型文件的耗时曲线图，如图 14 所示。

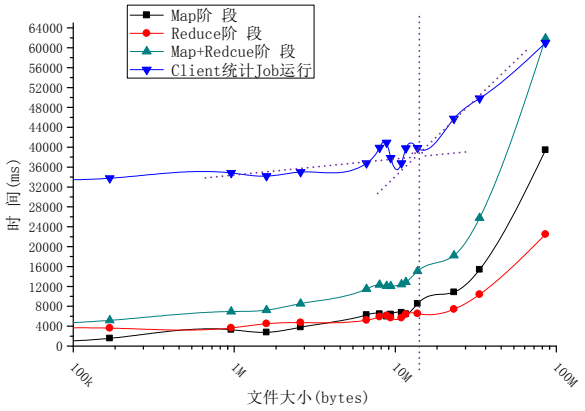

图 14 Job 的各个阶段随模型文件大小的增加耗时的曲线图

从图 14 中可以看出在文件小于 10MB 时模型文件处理的时间都普遍小于 40 秒，而大于 10MB 的文件的处理时间明显增加，所以本文将 10MB 作为小文件与大文件的界线。

4.4 MapReduce 形状分布算法实验与分析

本文实现了使用 MapReduce 编程思想所设计的形状分布算法，并在实验室搭建的 Hadoop 环境进行了实验，记录了运行时间、输入输出数据大小、Map 数和 Reduce 数，并对作业的运行效率进行了分析与总结。

运行参数设置为可调，其中采样点数 sampleNumber，默认为 1000；统计采样点的统计分布直方图区间数 binNumber，默认为 2000；特征向量的维数 featureNumber，默认为 2000。

本实验测试了模型数目分别为 11、100、1000、10000 的子测试集的预处理、采样、特征提取的 Job，每个步骤都是一个 Job，总共 12 个 Job，统计了运行数据如表 2 所示。

表 2 MapReduce 形状分布算法的实验对比数据

执行阶段	记录指标	11 个模型	100 个模型	1000 个模型	10000 个模型
Job1 模型预处理	Map 数	11	100	1000	10000
	Reduce 数	9	9	9	100
	输入文件大小（bytes）	2,055,426	31,637,951	444,506,289	3,013,674,523
	输出文件大小（bytes）	18,756,489	1,324,357,979	80,551,713,082	302,509,496,009
	运行时间（秒）	57	160	2299	2029
Job2 随机采样	Map 数	9	9	604	2295
	Reduce 数	5	10	10	100
	输入文件大小（bytes）	18,756,489	1,324,357,979	80,551,713,082	302,518,694,548
	输出文件大小（bytes）	1,220,200	17,396,469	39,037,197	68,310,091
	运行时间（秒）	73	160	830	2294
Job3 距离计算、分布统计、函数拟合、特征向量采集	Map 数	5	10	80	100
	Reduce 数	1	10	10	20
	输入文件大小（bytes）	1,220,855	17,396,469	39,037,197	68,323,491
	输出文件大小（bytes）	225,179	3,128,910	11,494,451	10,134,728
	运行时间（秒）	90	70	159	159
总运行时间（秒）		220	390	3,288	4,482
单机运行（秒）		13	190	4,804	56,603

从表 2 中可以看出，在数据量方面，随着模型数目的增多，需要处理的数据量越来越大，MapReduce 过程中产生的中间结果也越来越多，10,000 个模型在 Job1 后产生了 302GB 的数据。经过采样、特征提取后，输出的数据量开始变小，Job3 的输出数据只需要经过简单的格式化处理就可以为三维模型的检索提供服务。

从 Map 和 Reduce 数目来看，随着模型数量的增

多，需要的 Map 数目明显增加。而 Reduce 数是作为参数写入程序中的，如果处理数据量大，随着 Reduce 数的增加，job 运行的时间会明显缩短，如 job2 中的数据，100 个模型和 1000 个模型的 Reduce 数相同但是 1000 个模型的数据量大处理的时间过长，在优化程序之前为 40 分钟，而 10000 个模型的测试由于增加了 Reduce 数目到 100，运行时间小于 40 分钟。

从运行时间看，Map 阶段处理业务简单，而 Reduce 阶段业务复杂需要多次迭代，占整个 Job 的大部分时间。随着数据量增加，运行时间也增加，但是模型数量呈 10 倍增加时，运行时间增加幅度较小。

本文为了分析模型特征提取的三个 Job 在 Hadoop 集群上的运行情况给出如下定义，

Hadoop 集群资源使用量：

$$R = t * n \qquad （式 3）$$

其中 t 为程序运行时间，n 为程序占用 Reduce 数目；

单位模型使用集群资源量：

$$r = \frac{R}{f} \qquad （式 4）$$

其中 f 为模型数量。

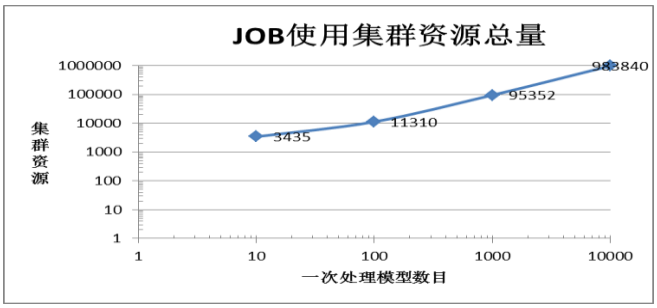

图 15 单次模型处理模型数目与集群资源使用量的关系图

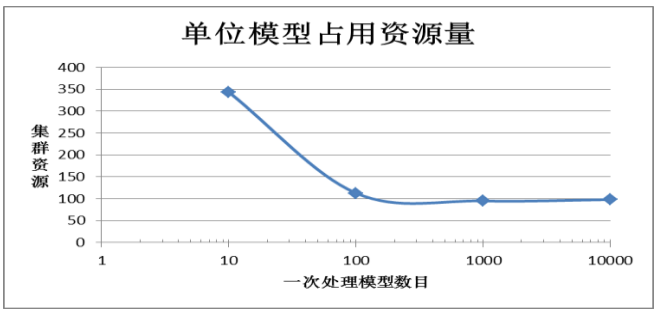

图 16 单次模型处理模型数目与集群资源使用量的关系图

图 15 为单次模型处理模型数目与集群资源使用量的关系图，可以看出随着模型数量的增多，占用集群的资源量呈线性增加。图 16 为单次模型处理模型数目与集群资源使用量的关系图，单次处理模型数量小于 100 时，由于需要进行 Job 运行的准备时间，单位模型占用资源量比较大，然而随着单次处理模型数量的增加，Job 准备阶段只需要 3 次，单位模型占用资源量趋于稳定。

从表 3 可以看出，使用并行化集群的处理时间和在单个计算机节点上的处理时间的对比，在数据量比较小时（小于 100 个模型时），由于不需要准备 Job 所需资源，单机串行处理模型时间比使用 MapReduce 时间短，但是当模型达到一定数量（大于 1000 个模型）使用 MapReduce 与使用单机串行处理相比效率明显提升。在拥有海量三维模型文件的三维模型检索系统中使用 MapReduce 来对模型进行批处理可保证系统在提供服务时的时效性。

4.5 MapReduce 基本性能优化策略

经过多次 MapReduce 处理三维模型实验，本文总结出以下策略优化处理三维模型的 MapReduce 程序。

（1） 重用 Writable 类型

在一个 map/reduce 方法中为每个输出都创建 Writable 对象，会导致程序分配出成千上万个短周期的对象，这些对象需要程序执行环境的垃圾收集机制来回收，影响 job 运行，若分配的对象过多可能会出现内存溢出等异常。本文将输出的 Writable 对象作为 Mapper 类或者 Reducer 类的成员属性，通过 Writable 对象的 set 方法为其赋值然后输出，避免短 Writable 对象的重复创建。

（2）避免正则表达式的重复编译

在模型文件处理各个顶点位置时用到了正则表达式。如顶点 v $(\mathrm{v} \quad x_i \quad y_i \quad z_i)$ 的正则表达式：

$$/\mathrm{v}\left(+\left[\backslash d|\backslash.|\backslash -|e\right]+\right)\left(\left[\backslash d|\backslash.|\backslash -|e\right]+\right)\left(\left[\backslash d|\backslash.|\backslash -|e\right]+\right)/g \qquad （式 5）$$

正则表达式编译过程是一个很耗时的过程，map/reduce 重复编译正则表达式会延长作业运行时间，严重影响集群性能，本小节末尾将通过一组实验来验证正则表达式的重复编译对 Map 性能的影响。本文令匹配模型顶点信息的正则表达式在 Mapper 加载时编译得到 Pattern 对象，以后重复使用该对象，避免了正则表达式重复编译降低作业运行时间。

（3）合理调整 Map 数和 Reduce 数

Map 数目一般由 Split 决定，但是当文件过大，或者很小的文件生成极大量的数据时，需要设置 mapred.max.split.size 参数来调整最大分片，并适当增加 Map 数，使用集群的力量解决中间数据生成过多的问题。而 Reduce 数目是需要人为设定的，如表 3 当只对 10 个模型进行预处理时，Reduce 数设置为 9，使得最后的输出文件有 4 个为空，而这个任务却占有这 4 个 Reduce 的运算时间。

（4）用记录行替换对象

Map 与 Reduce 间的数据传输是耗时的过程，这个过程包含了 Map 阶段向磁盘写数据，以及 Reduce 阶段从磁盘读取数据。把模型的变换处理的各个操作分别进行处理，需要至少三个作业先后运行。本文为了减少两个阶段的中间数据产生，把对模型的标准化

进行统一处理，在 Map 阶段对每行记录，即模型的顶点信息和面片信息先后做平移、旋转、缩放的向量变换处理，节省了大量的时间。

为了确定正则表达式的一次编译和重复编译对 Map 时间的影响，本文进行了一组对比试验。在对三维模型的标准化过程中，先不做任何优化进行试验，即正则表达式编译多次（编译次数与模型文件记录条数相等）；然后，对程序做出优化，正则表达式只编译一次，分别记录两次试验对作业运行时间的影响。本文从测试集中挑选不同的模型文件，组成总文件大小呈递增趋势的 15 个模型子测试集，进行上述两次实验，实验结果如图 17。通过对选为样本的 15 个模型子测试集处理的时间进行分析可知正则表达式重复编译大约可占到模型处理作业的 5%-12%。可见本文对所有匹配模型的正则表达式编译一次，避免大数据量三维模型边和顶点信息的重复编译，使 Hadoop 集群处理三维模型的时间明显缩短。

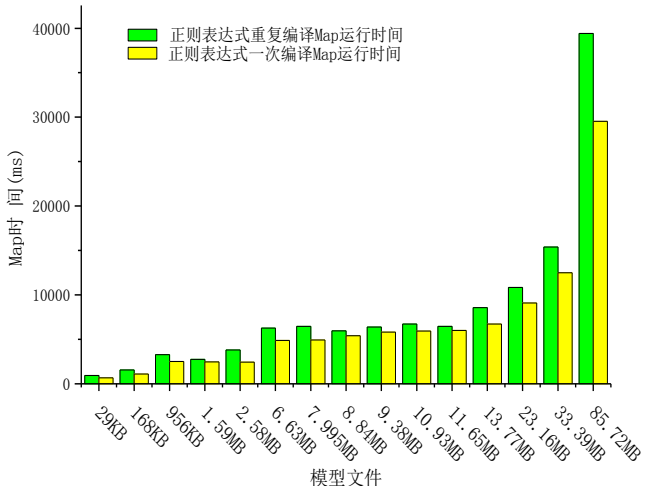

图 17 正则表达式重复编译与编译一次对 Job 的影响

5 结论与展望

为了应对三维模型数量的急剧增长给模型存储和处理带来的问题，本文采用 Hadoop 的分布式文件系统 HDFS 存储海量的三维模型，提出了使用模型附带信息，进行基于语义的相似度分类，然后通过装箱算法实现了有文件大小差异三维模型文件的合理存储。为了处理海量的三维模型数据，本文实现了 MapReduce 对三维模型标准化的处理操作，并与传统的三维模型形状分布特征提取算法相结合，提高了模型处理阶段的吞吐量，并为处理模型的 MapReduce 程序进行了性能优化。下一步工作是开展 MapReduce 环境下的三维模型检索研究，尝试将新的特征提取算法与 MapReduce 进行结合，在提高时间效率的同时，保证检索的准确度。

参 考 文 献

[1] Princeton 3D Model Search Engine[EB/OL].http://shape.cs.princeton.edu/search.html.
[2] Purdue 3D Shape Benchmark[EB/OL]. http://engineering.purdue.edu/precise/PBenchMark.html
[3] 3D Models Search Engine [EB/OL]. http://3d.csie.ntu.edu.tw/~dynamic/cgi-bin/DatabaseII_v1.8/index.html
[4] Sanjay Ghemawat, Howard Gobioff, Shun-Tak LeungGoogle. The Google file system [J]. ACM SIGOPS Operating Systems Review - SOSP '03 Homepage, 2003, v37 (5): 29-34.
[5] Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters [J]. Communications of the ACM, 2008, v51(1):107-113
[6] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler. The Hadoop Distributed File System , Proceeding of Mass Storage Systems and Technologies (MSST), 2010, 1-10
[7] White T. Hadoop: the definitive guide[M]. Yahoo Press,2010.
[8] LI Jian-jiang1,CUI Jian1,WANG Dan1,YAN Lin1,HUANG Yi-shuang2. Survey of Map Reduce Parallel Programming Model. Acta Electronica Sinica,2011, v39(11): 2635-2642 (In Chinese)
(李建江,崔健, 王聃, 严林, 黄义双. MapReduce 并行编程模型研究综述. 电子学报 ,2011,v39(11):2635-2642)
[9] 阿里云渲染[EB/OL].[2014-01-10] http://render.aliyun.com/
[10] R. Osada, T. Funkhouser, B. Chazells, D. Dobkin. Matching 3D Models with Shape Distributions[C], Proceedings of international conference on shape modeling and applications, Genova, Italy, 2001,154-167.
[11] Liu X, Han J, Zhong Y, Han C. Implementing webgis on hadoop: a case study of improving small file i/o performance on hdfs. In: IEEE international conference on cluster computing and workshops, 2009. CLUSTER'09, 2009. p. 1–8
[12] Bo Dong, Qinghua Zheng, Feng Tian, Kuo-Ming Chao, Rui Ma, Rachid Anane. An optimized approach for storing and accessing small files on cloud storage [J]. Journal of Network and Computer Applications, 2012, 35: 1847-1862.
[13] Grant Mackey, Saba Sehrish, Jun Wang. Improving Metadata Management for Small Files in HDFS[C]. Cluster Computing and Workshops(1552-5244), 2009, 1-4
[14] Fu S, Huang C, He L, et al. iFlatLFS: Performance optimization for accessing massive small files[C]//High Performance Computing (HiPC), 2013 20th International Conference on. IEEE, 2013: 10-19.
[15] Princeton Shape Retrieval and Analysis Group. Princeton Shape Benchmark [EB/OL]. http://shape.cs.princeton.edu/benchmark/
[16] 彭京，杨冬青，唐世渭，等. 基于概念相似度的文本相似计算[J]. 中国科学 F 辑：信息科学，2009，39(5)：534-544
[17] The Stanford Natural Language Processing Group [EB/OL].http://nlp.stanford.edu/index.shtml

李海生，男，1974 年生，博士，教授，中国计算机学会 CCF 高级会员，主要研究方向为计算机图形学，E-mail: lihsh@th.btbu.edu.cn。

赖龙，男，1988 年生，硕士研究生，主要研究方向为计算机图形学。

蔡强，男，1969 年生，博士，教授，主要研究方向为科学可视化。

毛典辉，男，1979 年生，博士，副教授，主要研究方向为大数据处理。

陈谊，女，1963 年生，博士，教授，主要研究方向为虚拟现实与多媒体。