低熵图像序列无损压缩

汤 颖[1] 刘晓哲[1] 张宏鑫[2]
(浙江工业大学计算机科学与技术学院 杭州 310012)[1]
(浙江大学CAD&CG国家重点实验室 杭州 310058)[2]

摘 要 大规模的云渲染技术带来了大量的三维图形渲染数据。为了减小集群渲染产生的图像序列数据的传输以及存储代价,针对渲染图像序列低熵的特点,基于字典编码技术提出了降低数据局部复杂性的无损数据压缩方案。该方案通过数据重排技术来大大提高数据的局部冗余度,从而提高数据无损压缩效率。为了进一步解决大规模图像序列的压缩耗时问题,提出了一种云计算平台上的分布式图像压缩处理方案,充分利用现有云计算中Map/Reduce计算模型实现了分布式编码方案。实验结果证明,对于渲染产生的大规模低熵图像序列,提出的方案能够有效提高编码率并减少编码时间。

关键词 LZ77压缩方法,图像序列,无损数据压缩,云计算
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.12.052

Low Entropy Image Sequences Lossless Compression

TANG Ying[1] LIU Xiao-zhe[1] ZHANG Hong-xin[2]
(School of Computer Science and Technology, Zhejiang University of Industry, Hangzhou 310012, China)[1]
(State Key Lab of CAD&CG, Zhejiang University, Hangzhou 310058, China)[2]

Abstract A large-scale amount of 3D graphic rendering image data has been brought because of the cloud rendering system. In order to reduce I/O transmission and storage cost of cluster rendered image sequences, this paper presented a lossless compression scheme based on dictionary technology which can get more compression by decreasing local complexity of data. A data rearrangement technology is applied to increase the degree of local redundancy, which can get more dense compression. To further improve the compression performance of a large-scale number of image sequences, the paper proposed a distributed image compression scheme based on the cloud computing infrastructure. The method was realized according to the current Map/Reduce computing model. The experimental results show that the approach proposed in this paper can make the coding process more efficiency.

Keywords LZ77, Image sequence, Losslessdata compression, Cloud computing

1 引言

计算机图形学的三维场景渲染是一项典型的计算密集型任务。随着云计算技术的兴起与发展,学术界与工业界纷纷提出了基于云计算架构的渲染农场解决方案——云渲染技术[1]。例如,浙江大学与阿里巴巴公司于2010年开始联合研发在线云渲染系统[2]。这一构想的提出与实现,使得渲染过程中的计算瓶颈得到了有效的解决。

大规模的渲染计算带来了数据量的急剧增长,以阿里云平台渲染的3D电影《昆塔传奇》为例,单个视角132帧的中间数据高达10GB,同样在电影《阿凡达》渲染制作过程中每秒的数据处理量也高达8GB。为了加快渲染速度以及便于后期修改,通常会采取分层渲染技术,因而渲染会产生不同的分层图像序列。分层图像序列帧内具有极大的相似性,这意味着它蕴含较少的信息熵,因此渲染产生的图像序列是一种低熵数据。

按需付费[3]的云商业模式使得渲染可以作为服务租赁,而且产生的原始图像序列需要在后续合成步骤中进行叠加与编辑,因此从艺术家的需求来说,他们不希望压缩过程中存在数据丢失。已有的图像编码技术如JPEG、LBG矢量量化编码通常是有损压缩方案。视频无损编码方案虽然能够取得高的压缩比,但同样存在精度损耗等问题,很难达到绝对无损。通用的无损编码方案,如霍夫曼编码、字典编码,由于未充分考虑序列数据之间相似性较大的特点,因此压缩效率也不高。

为了解决渲染图像序列无损编码效率偏低的问题,文中分析了字典技术中数据的压缩效率和局部复杂性之间的关

到稿日期:2013-06-25 返修日期:2013-08-16 本文受国家青年科学基金项目:网络环境下的大规模纹理数据压缩传输技术研究(61003265),国家自然科学基金:基于形状文法和多源数据融合的三维建筑高效重构方法研究(61070073),浙江省创新团队子项目:云计算环境下的功能构件管理关键技术研究(2009R50009)资助。

汤 颖(1977—),女,博士,副教授,主要研究领域为计算机图形学、云计算等,E-mail:ytang@zjut.edu.cn;刘晓哲(1987—),男,硕士,主要研究领域为网络环境下的三维图形绘制;张宏鑫(1975—),男,博士,教授,主要研究领域为计算机图形学、云计算。

系；对于渲染产生的低熵图像序列，提出了可以通过减少数据的局部复杂性的数据块重排方案，从而提高压缩性能。该方案能够在保持编码时间基本不变的前提下，使得表示每个像素的平均比特长度显著下降。为了进一步加快编码速度，在考虑损失一定压缩比的前提下提出了一种在云平台上基于Map/Reduce计算模型[4]的分布式编码方案，该方案能够在保持相对高压缩比的前提下灵活地设置参与计算的Map进程数目，并且能够降低编码的处理时间。

本文第2节介绍了图像序列无损编码方案；第3节描述了低熵图像序列的块重排算法以及基于云计算的分布式处理方案；第4节是实验结果的展示和分析。

2 相关工作

渲染图像序列编码的相关工作主要与两个方面相关：1）图像/视频的编码；2）通用数据的压缩。

2.1 图像/视频编码

图像编码方案往往是利用统计特性及人的视觉特性对分解后的图像信号进行一系列有损处理，常见的有基于分块DCT变换的JPEG编码、依据聚类思想的矢量量化编码以及基于小波变换的EZW编码[5]。这类基于变换的压缩方案往往能够取得高的压缩比，但是它们不是无损压缩，会损失一定的信息量。常用的无损图像编码方案有JPEG-LS以及TIFF、PNG等格式，基于预测的JPEG无损压缩模式虽然能取得好的压缩比，但压缩速度较慢且只适合于连续色调图像。JPEG-LS改善了JPEG无损压缩模式中的预测方案[6]，但是相对大规模数据，压缩效率还是较低。渲染系统产生的图像序列具有极大的时间冗余性，可以看作是加入了时间冗余的图像序列，而一般的图像无损编码方案没有考虑时间冗余，会导致压缩效率偏低。

视频压缩考虑了时间冗余，通过运动预测与补偿技术能够极大地提高压缩比，有损视频编码方案中因存在量化步骤会丢失数据，比如广泛应用的H264协议[7]。视频无损编码方案对关键帧和差分帧采取预测方案编码。HUFFYUV[8]对帧间的差值进行霍夫曼编码，可以压缩RGB和YCbCr颜色空间格式并且支持随机读写。AVIzlib未考虑帧间冗余，对每一帧通过zlib编码，能达到绝对无损，但效率偏低。MSU是一种空间有效的无损视频编码方案，能够对3个颜色空间无损压缩[8]。考虑到视频无损编码需求，H264扩展部分提供了一种无损方案Fidelity Range Extensions(FREXt)，该方法抛弃了H264变换和量化步骤，帧内采用了大量的空间预测，较其它静止图像编码方案性能有很大的改善。针对FREXt帧内预测计算复杂的问题，一种精简的帧内预测模型[9]通过损失小部分的压缩空间来达到快速计算的效果。FREXt使用的是YCgCo颜色空间，事实上并不能达到绝对无损。CorePNG对关键帧和差分帧分别采用绝对无损的PNG格式编码[8]。

云计算的发展为解决大规模数据处理和存储提供了很好的平台，尤其对于大规模且要求实时性的媒体流。Hadoop平台下的视频转码方案通过采用在Mapper端进行转码，在Reducer端进行视频合并的策略以达到节约时间的目的[10]，对于大规模图像编码可以通过云上图像处理平台实现多通道多任务并行处理。

2.2 通用数据编码技术

通用数据编码方案不考虑数据的具体格式，且在编码过程中不丢失信息。根据信源数据的分布规律特性，通用数据编码大致可以分为两类：第一类是基于数据统计特征的编码算法，如常见的霍夫曼编码、行程编码以及算术编码等，这类算法认为要编码的信源符号之间是相互独立的，而且通常需要预先计算数据的概率分布或者其它统计特征，因而应用受到了一定的限制；另一类是应用广泛的字典技术，编码过程不需要预先知道数据的统计特性，利用数据的局部重复特性来取得好的压缩效果，实现简单且具有很强的自适应性。下文主要介绍与本文相关的字典压缩技术。

LZ77和LZ78[11,12]字典压缩技术奠定了字典编码技术的基础，并且其压缩比接近随机序列的信息熵。LZ77技术从已经编码的数据流中寻找匹配，维护了一个显式的滑动窗口机制，对在滑动窗口中寻找到的匹配串用索引替代。其中应用最广泛的基于LZ77算法的变异是Phil Katz的DEFLATE[13]方案，该方案是一种混合编码机制，被应用在gzip算法以及png图像格式中。LZMA同样是LZ77算法的变异，它扩大了字符串匹配的搜索范围，因而压缩比有所提高。另一方面，Burrows等人[14]从数据分块压缩角度出发，提出一种块排序的无损压缩算法，该算法通过对数据块进行可逆的矩阵变换后使得数据更容易压缩，但该方法存在处理速度过慢的问题，bzip为该算法的开源实现。

LZ78系列构建了一个显式的字典，该算法在编码端和解码端能够动态地自适应构造相同的字典，并且避免了对已经编码序列的依赖。LZW[15]算法减少了LZ78算法需编码的数据元组数，提高了处理速度，该算法被应用在GIF图像无损编码中。从信息论角度考虑，压缩算法的性能很大程度上受制于对信源先验知识的获取。对特定信源知识的获取有助于我们针对特定的信源提出具有针对性的专用压缩方案。本文提出的针对低熵渲染图像序列的压缩方法即利用了渲染数据高冗余性的特点，基于冗余性分析的渲染数据重排有助于减少数据的局部复杂性，从而更好地利用字典编码技术中的局部模式匹配。实验结果证明该方法能够提高压缩比。

另一方面，图像视频等多媒体内容的数据量巨大，也会导致压缩以及解压缩的时间成为系统的瓶颈。快速编码方案既考虑了系统的性能，又照顾到了IO的吞吐能力，在大规模数据存储与分析领域得到了广泛应用。针对LZ系列算法字符串匹配比较耗时的问题，Ross N. Williams提出了一种快速编码方案LZRW1[16]，该方案通过构建一个哈希表来加快子串的匹配。当前快速编码主要有Google公司推出的snappy[17]，QuickLZ[18]和FastLZ[19]等。快速编码方法通常没有熵编码步骤，Gipfeli[20]采用了类似snappy的机制并且增加了熵编码过程，在保持快速编码的同时提高了压缩比。LZ4[21]适合大吞吐量的压缩场合，它采取和CPU缓存相结合的机制来提高处理速度。

3 基于数据块排序的图像序列无损压缩方案

3.1 低熵图像序列数据块重排编码

渲染产生的图像序列分层内具有极大的全局冗余信息，

不同的分层之间同样也可能存在较大的冗余信息，例如立体视频中从不同角度观察产生的图像序列。冗余量和序列图像内部的相似程度成正比，序列之间的相似性越大，冗余信息就越多。数据的信息量可以用熵度量，数据的重复性越大，信息熵越低，从这一方面考虑，渲染图像序列分层存在时间和内容上的全局冗余，这是一种典型的低熵数据。极端情况下，当渲染的图像序列完全相同时，整个图像序列的信息量基本等同于单帧图像的熵值。

图 1 为 LZ77 滑动窗口机制数据编码原理示意图，LZ77 算法基于数据分布存在局部性原理，认为过去最近的数据流会重复出现。图示中的匹配缓冲区表示最近已经编码的数据，即大小为 W 个字符的滑动窗口区域，搜索区为将要编码的数据流。对搜索区域的数据在滑动窗口区域内进行最长字符串匹配，其中区域①、②为在匹配缓冲区取得的最长匹配，编码对搜索区中取得匹配的字符串用长度 L 和偏移量 O 替换，从而达到减少数据的目的。从 LZ77 的滑动窗口机制可知，数据的压缩比与局部范围内的子串重复程度有关，局部范围子串的重复程度越大，则匹配命中率越高，因而压缩越紧密。

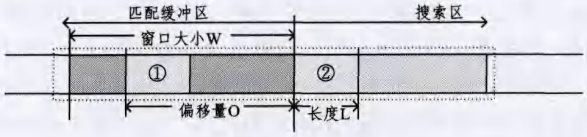

图 1　LZ77 滑动窗口机制

定义 $S_n(n\geqslant 0)$ 为缓冲区中将要编码的数据和字符串模式匹配所覆盖的区域，C_n 为 S_n 编码后的输出结果，存在映射 $f:S\rightarrow C$ 使得

$$C_i=f(S_i,S_{i-1}) \tag{1}$$

定义子串 S_i 的信息量为 $H(i)$，输出 C_i 的长度为 $L(C_i)$，字符串 S 的压缩效率为 η，那么

$$\eta=\sum_{i=1}^{n}L(f(S_i,S_{i-1}))/L(S) \tag{2}$$

无损编码方案的效率通常可以通过压缩比和处理时间来度量。压缩比 $\lambda=1/\eta$，其中 λ 为原始数据的大小和编码后数据大小的比值，λ 取得的值越大表明数据压缩得越紧密。显然对于给定大小的匹配缓冲区，压缩的效率 η 取决于对子串的划分，低复杂度的子串意味着蕴含更少的信息量，子串的内容相关性越好，匹配速度和压缩的紧密程度越能得到提升。增加匹配缓冲区的大小能够提高匹配的命中率，因而能够提高压缩比，但会造成子串的匹配速度较慢，从而导致压缩时间偏长。另一方面，基于 LZ78 的 LZW 算法中，编码的输出长度同样与字典的局部命中率相关。

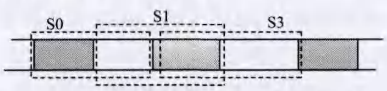

图 2　编码区域滑动状态依赖关系示意

举例说明如图 2 所示的编码过程中缓冲区维持的状态依赖关系，假设存在 3 个内容相关的子串 $S_0S_1S_2$，其中 S_0 = “abcabb”，S_1 = “abcabc”，S_2 = “abcbbb”，待编码的搜索区和已经编码的匹配缓冲区大小都为 3 个字符。为了使得编码效率最高即得到式(2)的最优解，可以通过降低 S_i 的复杂度即寻找一种信息熵较少的子串来提高编码效率。在信源统计分布特性已知的情况下，我们可以针对不同的数据采取降低子串复杂度的方法来改善压缩性能。比如对于字符串 $S_0S_1S_2$，由于已知其信源分布，采取最简单的排序方法得到 S = “abcabcabcabbabcbbb”，并对其进行编码，很显然，对 S 的压缩效率要显著高于分别对 S_0S_1 和 S_1 的处理。同样分析基于 LZ78 技术的 LZW 算法可知，编码的输出和该算法维持的字典更新频率存在一一对应关系[22]，减少局部子串的复杂度能够提高字典的命中率也即意味着字典能够保持持续收敛，字典收敛越快，编码输出就越少，进而数据的压缩比也就越高。

对图像序列做无损压缩大致可以从两个角度考虑：一种是考虑了图像内部数据冗余性的静止图像编码方案，该方案未利用帧间冗余，当图像数据量较大时压缩时间以及整体压缩率都会比较低；另一类方案采用基于预测的编码方案，分别针对帧间时间冗余和帧内空间冗余进行预测，这类方法压缩比较高，但往往计算比较复杂，而且对于存在颜色空间转换的编码算法也很难达到绝对无损。基于减少数据的局部复杂度以及充分利用帧间冗余的思想，我们提出了一种结合以上两种方案的混合策略——低熵图像序列的数据块重新排序算法。

定义长度为 n 帧的输入图像序列表示为集合

$$I=\{P_1,P_1,\cdots,P_i,\cdots,P_n\}(1\leqslant i\leqslant n)$$

并且 P_α 相似于 $P_\beta(1\leqslant\alpha\leqslant n,1\leqslant\beta\leqslant n,\alpha\neq\beta)$，集合 I 中对应的图像帧表示为集合

$$P_i=\{D_{i1},D_{i2},\cdots,D_{ij},\cdots,D_{ik}\}(1\leqslant j\leqslant k,1\leqslant i\leqslant n)$$

其中，P_i 中的元素 D_{ij} 为对应图像帧的数据块，k 表示对应图像帧的分块数，则经过数据块重排后的输出表示为集合

$$Q=\{F_1,F_2,F_i,\cdots,F_m\}(1\leqslant m\leqslant n)$$

其中，$F_i=\{D_{1i},D_{2i},\cdots,D_{ni}\}(1\leqslant i\leqslant m)$，并且存在恒等变形 $I\equiv Q$。由式(2)可知，对于给定有限长度序列数据，压缩率取决于对应分块数据的局部复杂性，而数据的局部复杂性可以通过数据分块内部和邻近数据分块之间的相似程度来表示。经过变换之后，数据的局部复杂特性得到了有效的降低，编码过程是对数据 Q 进行处理。

图 3 为图像序列数据块重排示意图。对于给定渲染图像序列，由于整个序列的长度比较短，而且图像序列中每一帧的数据量高达十兆字节以上，不同帧之间相同位置的像素值具有极大的相似性。为了有效地利用数据块重排策略，实际操作过程中我们将图像序列作为二进制数据流处理，这样虽然损失了一部分压缩比，但处理起来非常方便。基于字典技术的数据编码过程是一种流式处理方案，数据源源不断地读入缓存区后通过编码输出。

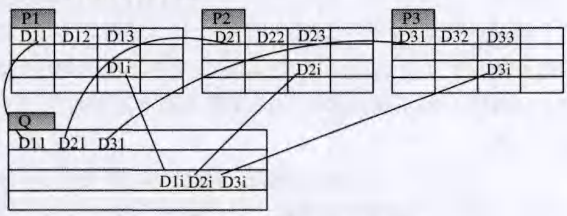

图 3　图像序列数据块重排原理图

基于字典的块重排编码方案可以分为 3 个步骤。首先是计算图像序列中每一帧的分块数，数据分块越多会导致处理

的时间增长，每一帧分块数目由设置的数据重排块大小决定。然后，循环依次读取图像序列中每一帧的数据分块并将读取的数据块拼接为局部复杂性较低的文件。最后对低复杂度数据采取基于字典的编码方案进行压缩。另一方面，块大小的划分直接影响系统的压缩效率，重排块大小的选取主要受字符串的匹配范围、图像序列的长度和大小以及分辨率、图像序列之间的相关性等多方面因素影响。表 1 为块排序算法的伪代码。

表 1　图像序列块排序策略伪代码

```
sum：图像序列帧长度
per_count[sum]：每一帧对应分块数目
while(sum){
tags：标记图像序列帧长度
    while(tags){
        读取当前图像帧对应的数据块；
        调整当前帧对应分块数，移动指针；
        if(当前帧对应数据分块数存在){
            sum--；
        }
        将读取的数据块写入到输出文件中；
        tag--；
        }
    重置 tag 为图像序列帧长度；
    调整文件指针位置；
}
对输出文件编码；
```

3.2　云上的低熵图像序列分布式编码

传统的基于字典技术的编码方案由于内存大小的限制，使得数据不可能全部存放到内存中，同时对处理时间的要求限制了系统字符串匹配的范围，因此不适合处理较大规模数据。快速编码通过牺牲压缩比来换取时间效率。云计算提供了一种处理大规模数据的机制。分布式编码方案由于没有考虑各个分片之间的冗余信息，会损失压缩空间，因此在允许损失一部分压缩空间的前提下可以通过云计算平台部署一种多通道的并行编码方案，该方案能够改善计算资源不足导致的编码处理时间过长问题。

Google 公司提出的分布式计算框架 Map/Reduce 以及分布式文件管理系统 GFS[23] 为大规模数据处理开辟了一种新的思路。云计算提供一种计算模型，它能够对共享的可配置计算资源按需提供一种方便的访问策略，具有很好的灵活性。传统观念认为云计算模型可以分为 3 个层次：软件层、平台层和基础设施层[24]。视频图像等多媒体数据包含大量的信息且数据量庞大，而且大规模视频图像处理往往对计算性能的要求也较高，即便是高性能超级计算机也很难满足大量视频图像等多媒体数据处理对计算量的需求。云计算平台海量的计算和存储能力能够解决多媒体数据处理中的存储和计算瓶颈。另一方面，由于视频或图像序列存在极大的时间冗余，编码可以认为是对一定长度的数据按照时间因素分段后分别单独处理，这种时间上的离散特性使得基于云平台的分布式编码方案具有可行性。云计算的可扩展性非常适合视频转码这类对计算量需求不稳定的应用。Pereira 等[25] 提出了一种适合视频压缩的 Split&Merge 处理模型，模型要解决的关键问题是如何对视频分段处理，合适的分段方法使其能够并行计算是减少整个系统处理时间的关键所在。

图像序列分布式编码方案采取了基于 Map/Reduce 的计算模型。Map/Reduce 通过将问题分发给多个计算节点处理然后合并结果集来解决单机性能不足的问题，主要包括两个阶段：Map 阶段中一个主节点从分布式存储上获取输入并将这些输入分解为小问题后分发给多个计算节点，这些计算节点各自独立处理；Reduce 阶段主节点收集计算节点中所有问题的输出并将结果合并。Hadoop Map/Reduce 是一个处理大数据集的计算框架，是 Map/Reduce 模型的开源实现，能够支持大规模的集群计算并具有可靠的冗错机制。HDFS 是 Google 分布式文件系统 GFS 的开源实现，要处理的数据通常存放在分布式文件系统 HDFS[26] 上。Hadoop streaming 提供了一种标准的输入输出接口，使得系统能够很容易地移植到其它编程语言中。

分布式编码系统要编码的图像序列存放在 HDFS 上，通过 Hadoop streaming 调用 Map/Reduce 计算框架来实现分布式编码。在调用分布式计算框架之前已经通过主节点对图像序列进行数据块重排，Map 阶段我们需要将排过序的图像序列分发给各个计算节点，各个计算节点负责对应数据块的编码。数据分发过程有两种方案：一种是根据 Map 节点数将排序后的数据预先分割为多个子文件，其中子文件的个数可以根据系统设定的 Map 节点数动态调整，然后将对应分割后的子文件分发给各个计算节点；另一种方案是将排序后的图像序列分别分发给各个计算节点，各个计算节点通过偏移量读取该节点负责压缩的序列分片数据，这种方案存在的问题是由于复制大量文件副本，会导致 IO 延迟。文中采取了先切分后分发的策略。

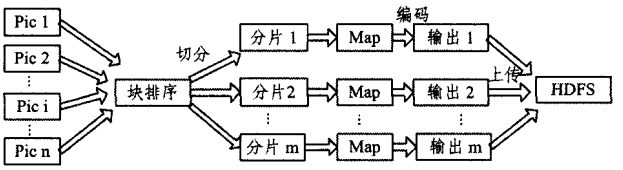

图 4　Map/Reduce 分布式图像编码原理图

多通道数据流编码过程处理方案如图 4 所示，在 Map 阶段之前对图像序列中图像帧 Pic 1 到 Pic n 运用数据块重排策略进行排序，然后将排序后的文件切分为分片 1 到分片 m 共 m 份，每个 Map 读取对应的分片文件后编码，最后将编码后的输出上传到分布式文件系统 HDFS 上。压缩可以认为是一个多 Map 无 Reduce 步骤的处理过程。解码过程需要将编码产生的 m 份输出数据分别下载到相应的计算结点，每个 Reduce 进程读取相应的输入分片，然后调用相应的解压缩算法，最后将解压后的分片数据上传到 Master 节点，其中 Master 节点根据块排序中保存的信息来恢复图像序列。解码过程没有数据块重排步骤，与编码类似，解码可以认为是一个单 Map 多 Reduce 的过程。整个系统编解码过程可以分为下面几个步骤：

a)初始化，对单个图像序列通过一定的重排策略选取合适的重排块大小后对序列图像排序；

b)根据 Map 数量设置分片个数，然后对 a)步骤产生的文件进行分割，调整各个分块的大小，为了简单，文中采取平均块大小分配策略；

c)将步骤 b)中的分块数据上传到分布式系统 HDFS 中，通过文本方式记录对应文件在分布式文件系统中的绝对地址；

d)分布式编码实现，调度集群实现并行处理，每一个计算节点首先从 HDFS 下载对应文件到本地，在本地对数据压缩后将文件上传到 HDFS 上；

e)分布式解码实现，根据编码后输出文件数目启动对应

数量的 Reduer，每个 Reducer 进程从 HDFS 中下载对应的已编码文件到本地，对数据解压缩后上传到分布式文件系统中；

f)从分布式文件系统中下载已经解码后的数据，根据步骤 a)保存的信息，恢复图像序列。

上述方案通过传输文件路径避免了直接对二进制数据的操作，预先分割的方法较传输偏移量也能在一定程度上减少 I/O 负担；另一方面，由于分布式文件系统分布在不同的计算节点上，因此也可以通过本地化减少机器之间的数据传输，从而节省带宽。

4 实验结果与分析

实验中 PC 单机的硬件配置：CPU 型号为 Intel(R) Core (TM)2Quad CPU Q6600@2.40GHz 2.39GHz，内存为 4GB (DDR2 SDRAM)。集群配置为 8 台服务器，CPU 型号为 Intel(R) Core(TM)i7-2600 CPU@3.40GHz，内存为 6GB。

三维渲染中用户通常将场景划分为多层进行渲染，比如背景层、色彩层以及立体视频中同一帧不同方向的图像分层。选取阿里云渲染服务不同场景产生的多个渲染图像序列进行压缩，图 5 中场景 A 为三维动画《昆塔传奇》中某一场景渲染产生的不同图像序列分层，分层图像的分辨率为 2048×871，色深为 24 位，每个图像序列帧数为 132 帧。图 5 表示场景 A 中的 3 个图像序列中每相隔 60 帧的数据，其中场景 A 中序列 1 的 EXR 格式数据高达 5.82GB，即便是 TGA 格式也达到 898MB；序列 2 中的 EXR 格式数据达到 3.01GB。图 6 中场景 B 为只有一个序列分层的 blender 渲染场景，图像的分辨率为 1920×1080，24 位色深，其中图像序列共 100 帧。场景 B 中的 TGA 格式数据达到 593MB 字节。图 7 中场景 C 为同样只有一个序列分层的 Maya 渲染场景中的一帧，图像的分辨率为 1024×1024，色深度为 24 位，24 帧该场景序列 TGA 格式文件达到 72MB。除此之外，实验另外选取了 21 个场景中的部分渲染数据进行了处理，其中每个场景共 50 帧，图像序列总的数据量高达 17.5GB。图 8 为 3 个不同的渲染场景中的特定分层数据帧。

图 5 场景 A 不同分层序列帧

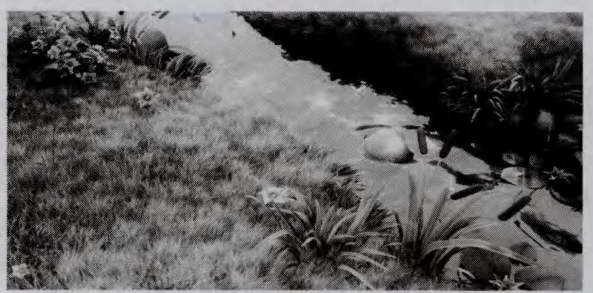

图 6 场景 B blender 渲染场景

图 7 场景 C Maya 渲染场景

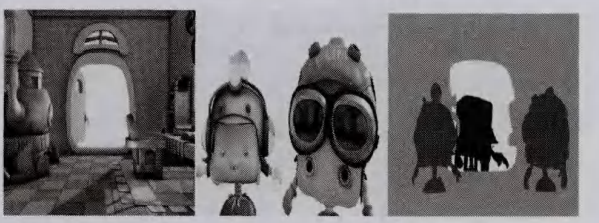

图 8 其它部分渲染场景分层帧

4.1 实验结果

图像序列采用基于块排序的无损压缩模式，定义图像序列之间的相关性为 $R(0<R<1)$，当 R 无限趋近于值 1 时，图像序列之间的相似性达到最大。场景 A 中的图像序列 1 是作为静态背景的环境遮罩层，在整个渲染图像序列的相邻帧之间只有少许变化，因而该层的相似性最大，也即意味着蕴含的信息量相对最少，所以能够取得的压缩效率最高。

另一方面，基于块重排的图像序列压缩方案中压缩效率还取决于数据块的大小选取和序列长度，相关性较大的图像序列选取合适长度的块重排后局部数据复杂度降低，因而压缩效果较好。实验中选取了 3 种不同的图片格式，分别为未压缩的 TGA 格式、经过 DEFLATE 编码的 PNG 格式以及运用离散余弦变换的 JPG 编码格式。图 9 为场景 A 中相关性 R 接近于 1 的背景层图像序列 1 对应不同压缩算法排序前后压缩比示意图，图中横坐标表示选取不同大小的排序数据块(单位为字节)，纵坐标表示压缩比。

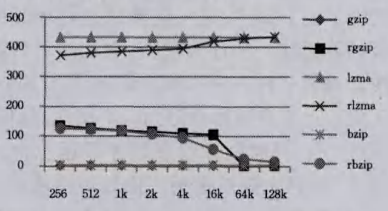

图 9 场景 A 序列 1 TGA 格式压缩性能比较

图 9 中 gzip、lzma 和 bzip 分别表示原始数据的压缩算法，与其对应的数据块排序后的压缩算法表示为 rgzip、rlzma 和 rbzip，后续图表中用同样方式表示。

图 10 和图 11 分别表示场景 A 中 PNG 格式以及 JPG 格式图像序列排序前后的压缩比示意图。从实验压缩比折线图中可以得出：对于图像序列间相似性较大的数据，重排方案都能提高压缩比，尤其对于未经过编码的 TGA 数据格式，重排后数据的局部复杂性显著下降，因而压缩比能得到极大的改善。因为原始场景 A 中序列 1 的 TGA 数据达到 898MB，而 lzma 算法的搜索窗口高达 4GB，所以图 9 中显示 rlzma 的压缩性能相对于 lzma 没有显著变化，而 rbzip 以及 rgzip 的压缩结果相对原始 bzip 和 gzip 显示压缩比能够提高 30 倍左右。

图 10 和图 11 中由于数据已经经过编码，其压缩效果的改善并没有 TGA 格式明显，但压缩比的提升依然可观。

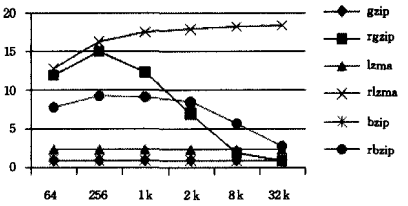

图 10　场景 A 序列 1 PNG 格式压缩性能比较

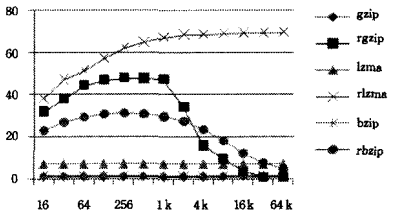

图 11　场景 A 序列 1 JPG 格式压缩性能比较

场景 B 和 C 中只有一层图像序列，因而相邻帧之间相关性比较小，压缩比的提高不如场景 1 显著，但通过选取合适的重排数据块仍然能提高压缩比。图 12 和图 13 为场景 B 和 C 分别选取 256 和 512 个字节大小的重排块前后压缩比的对较情况。在数据规模比较大的前提下，所能节省的空间还是比较可观的。

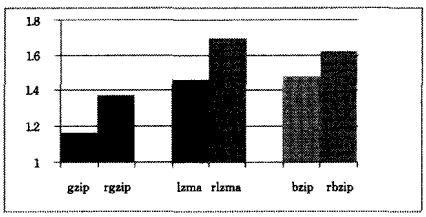

图 12　场景 B TGA 格式压缩性能比较

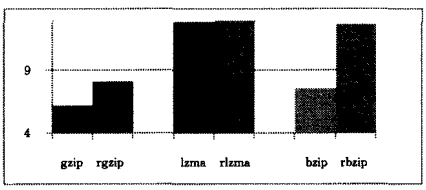

图 13　场景 C TGA 格式压缩性能比较

对于图像序列相关性 R 介于上述两者之间的场景 A 中的图像序列 2 和序列 3，同样可以根据序列特点选择重排策略来提高压缩性能。场景 A 中图像序列 3 中部分帧存在运动物体，导致序列相关性相对较低。在搜索缓冲区范围内通常选择小的数据分块往往能够获得好的压缩比，但处理时间会增长，而选取大的数据块能节约压缩时间，但会耗费更多的存储空间。表 2 为场景 A 中序列 3 JPG 格式下不同压缩方法的压缩性能对比，其中选取的重排数据块大小为 1024 个字节。

考虑到图像序列中可能存在运动物体(较之于静止元素，运动物体相邻帧之间存在较大的差异，从而使整个序列的相关性降低，导致压缩性能下降)，为了进一步提高压缩比，在基于数据块排序的压缩系统中如果能通过识别关键帧对序列分开压缩，性能会有进一步改善。表 3－表 5 为场景 A 中序列图像 3 去除相邻帧间变化比较剧烈的数据之后 3 种数据格式对应压缩性能比较。

表 2　场景 A 中序列 3 JPG 格式压缩性能比较

算法	压缩比	时间(s)
gzip	1.35	0.898
rgzip	1.61	0.761
bzip	1.49	8.346
rbzip	1.61	8.364
lzma	1.72	6.037
rlzma	1.74	5.597

表 3　场景 A 中序列 3 TGA 格式压缩性能比较

算法	压缩比	时间(s)
gzip	19.10	30.75
rgzip	34.09	28.09
bzip	50.42	194.21
rbzip	80.81	199.10
lzma	55.94	159.99
rlzma	67.78	137.41

表 4　场景 A 中序列 3 PNG 格式压缩性能比较

算法	压缩比	时间(s)
gzip	1.09	3.49
rgzip	1.11	3.49
bzip	1.19	14.85
rbzip	1.26	14.04
lzma	1.26	32.29
rlzma	1.29	31.08

表 5　场景 A 中序列 3 JPG 格式压缩性能比较

算法	压缩比	时间(s)
gzip	1.35	0.738
rgzip	1.66	0.667
bzip	1.51	7.413
rbzip	1.65	7.45
lzma	1.78	5.000
rlzma	1.80	4.813

从表 2 和表 5 的实验结果对比可知，去除序列图像中存在运动的图像帧能够一定程度上提高系统的压缩比。针对未编码的 TGA 格式的原始数据，表 3 的实验结果再次证明存在相关性的图像序列重排后压缩比的提升还是非常明显的。表 4 和表 5 中的数据由于已经经过编码压缩比，因此提高比较有限。从处理时间角度看，表 2 至表 5 的结果显示，对图像序列采取块重排策略之后再编码的方案，数据的处理时间基本没有太大的变化。

上述结果显示对于低熵图像序列，采取降低局部数据复杂性的方案能够有效地提高压缩比。该方法同时具有很好的自适应性，并且实现简单。我们可以根据不同的数据格式灵活设置重排数据块的大小，从而取得较好的压缩效果，达到节约空间的目的。要编码的图像序列占用的存储空间用 *sun* 个字节表示，*len* 表示帧的长度，其中每一帧图像的分辨率表示为 $m\times n$ 个像素，对应的颜色通道数目用 *channels* 表示，重排数据块大小的选取可以通过式(3)中的经验值来调整。其中重排数据块大小表示为 *block* 个字节，从而：

$$block=(1+\alpha)\times 2^{2\times(k+channels)}\times sum/(m\times n)\times l \qquad (3)$$

其中，$\alpha(0\leqslant\alpha\leqslant 1)$表示图像序列的自相关性，式(3)中设置 $\alpha\equiv 0$，k 表示正整数，其默认值为 1，用户可以通过调整 k 的值来获取优化的数据块。

4.2　并行编码实验结果

云平台上的分布式编码方案中，实验根据图像序列的大小来确定数据的切分数目。场景 A 中的序列 1 分层 EXR 数据格式达到 5.82GB，经过块排序之后设定将数据平均分割为 10 份然后分发给各个 Map 进程处理，各个节点的计算时间

以及压缩比统计结果如表 6 所列。为了便于比较，分别统计了顺序压缩所有分片和并行压缩所花费的时间，表中计算时间为各个计算节点对相应分块数据顺序编码所花费的时间总和，与之对应的压缩比为单机情况下对图像序列采取块重排编码方法后所能达到的比值。系统响应时间为编码任务从提交到计算结束整个系统在集群上的处理时间。从表 6 中的计算时间和系统响应时间对比可知，分布式方案在节约处理时间的同时还能维持较高的压缩比。

表 6　场景 A 中序列 1 Exr 格式压缩性能

	压缩时间(s)	压缩比
计算时间	349	41.5
系统响应时间	70	
Map1 rgzip	30	32.5
Map2 rgzip	51	32.5
Map3 rgzip	36	32.7
Map4 rgzip	30	37.2
Map5 rgzip	30	48.9
Map6 rgzip	27	46.6
Map7 rgzip	31	33.6
Map8 rgzip	33	42.2
Map9 rgzip	48	54.4
Map10 rgzip	33	50.6

表 7 是对场景 A 中序列 1 PNG 格式图像序列的处理结果统计。序列的数据量只有 238MB，因为数据量相对较小，采取将重排后的数据平均分割为 5 份的方案。PNG 格式的图像序列中每一帧都存在固定长度的头部数据，并且该数据块的相似性极大。表 7 中 Map1 节点压缩比相对其它节点较高，这是因为 Map1 节点处理的数据是相似性较大的序列帧中的头部数据。表格显示处理分布式处理方案的处理时间少于顺序编码每一个分块所耗费的计算时间。

表 7　场景 A 中序列 1 PNG 格式性能比较

	压缩时间(s)	压缩比
计算时间	48	12.3
系统响应时间	25	
Map1 rgzip	9	31.8
Map2 rgzip	9	11.4
Map3 rgzip	9	10.3
Map4 rgzip	9	10.4
Map5 rgzip	12	10.5

表 8 是对场景 B 中的 100 帧 TGA 格式图像序列压缩后的结果统计，实验过程中选取的重排块大小为 256 个字节，序列文件的大小达到 593MB，同样采取 5 个 Map 进程并行压缩。

表 8　场景 B 中 TGA 格式性能比较

	压缩时间(s)	压缩比
计算时间	129	1.38
系统响应时间	43	
Map1 rgzip	24	1.39
Map2 rgzip	30	1.35
Map3 rgzip	24	1.29
Map4 rgzip	27	1.23
Map5 rgzip	24	1.22

表 6—表 8 的结果表明，对于低熵的渲染图像序列分布式压缩方案仍然能够保持比较高的压缩比。另一方面，Map 节点数过多也会使得启动 Map 进程的开销增加。整个分布式编码方案采取的是基于 Hadoop Map/Reduce 的计算框架，通过 Hadoop Streaming 调用整个集群过程中需要先将消息通过 JNI 方式传递给调度系统，因而效率还能得到进一步的改善。但是相对于单机而言，分布式编码系统的响应时间还是远远少于单个节点顺序编码所有数据所花费的时间总和。当要处理的数据规模较大时，时间性能的改善还是比较可观的。同时基于云计算的分布式编码可以实现类似于视频转码的功能，并且不会显著增加系统负担。

结束语　依据图像序列的相似性以及 LZ77 系列字典压缩上的特性，文中针对渲染图像提出一种基于块重排的无损编码方案。该方案同时考虑了图像序列之间的冗余信息并且能够适应云计算分布式计算模型，较传统的 LZ 系列算法能够显著地改善压缩效率。同时为了进一步改善压缩速率，文中利用 Map/Reduce 分布式计算模型实现了适合块重排的图像序列无损压缩方案。另外计算节点数目的选取是影响系统性能的关键因素，在今后研究中需要着重考虑。

参 考 文 献

[1] Liu W, Gong B, Hu Y. A large-scale rendering system based on hadoop[C]// International Conference on Pervasive Computing and Applications. Port Elizabeth, 2011: 470-475

[2] http://render. aliyun. com

[3] Armbrust M, Fox A, Griffit R, et al. Above the clouds: aBerkeley view of cloud compu-ting. Electrical Engineering and Computer SciencesUniversity of California at Berkeley[R]. UCB/EEECS-2009-28. Technical Report, February 2009

[4] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[C]// Operating Systems Design and Implementation. 2004: 137-150

[5] Dhawan S. A review of image compression and comparison of its algorithms[J]. International journal of electronics & communicationtechnology, 2011, 2: 22-26

[6] Weinberger M, Seroussi G, Sapiro G. LOCO-I: a low complexity, context-based, lossless image compression algorithm[C]// Data Compression Conference. Snowbird, UT, 1996: 140-149

[7] Wiegand T, Sullivan G, Bjontegaard G, et al. Overview of the H. 264/AVC video coding standard[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2003, 13(7): 560-576

[8] Graphics and Media Lab Video Group, Computer Science, MSU. Lossless video codecs comparison[R]. 2007

[9] Milani S. Fast H. 264/AVC FRExt intra coding using belief propagation[J]. IEEE Transactions on Image Processing, 2011, 20(1): 121-131

[10] 杨帆，沈奇威. 分布式系统 Hadoop 平台的视频转码[J]. 计算机系统应用，2011，20(11)：80-85

[11] Ziv J, Lempel A. A universal algorithm for sequential data compression[J]. IEEE Transactions on Information Theory, 1977, 23(3): 337-343

[12] Ziv J, Lempel A. Compression of individual sequences via variable rate coding[J]. IEEE Transactions on Information Theory, 1978, 24(5): 530-536

[13] RFC 1951. DEFLATE compressed data format specification version 1. 3[S]

[14] Burrows M, Wheeler D J. A Block-sorting lossless data compression algorithm[R]. Palo Alto, California: digital systems research center, SRC Research Report, May 10, 1994

[15] Welch T. A technique for high-performance data compression [J]. IEEE Computer, 1984, 17(6): 8-19

（下转第 259 页）

利用分块特征收缩后的 HOG-LBP 特征对 INRIA 数据集中的行人测试图像进行行人检测，效果如图 4 所示。

图 4　行人检测示例

结束语　本文提出了一种基于分块特征收缩的行人检测的方法，该算法解决了传统特征向量维度高、漏检率高的问题。在公共数据集上通过大量样本进行测试，结果表明，本文所提出的方法比 HOG 特征向量的维度下降了约 1000 维，同时在误检率为 10^{-4} 时漏检率降低了约 2%；比起 LBP 特征向量的维度下降了约 3000 维，同时在误检率为 10^{-4} 时漏检率降低了约 2%。从图 4 中可以看出，本文所提出的方法对于复杂的背景、行人多种姿态、摄像机多视角、小部分遮挡、尺度变化等情况也具有一定的鲁棒性。

本文所提出的方法需要在行人检测之前对一小部分样本进行训练和测试，这样会花费一定的时间，这是本算法存在的不足之处。另外，本算法还无法用于行人大面积遮挡等复杂环境下的行人检测。

在未来的工作中，我们将从以下几个方面展开进一步的研究：1)将更多的特征进行融合。本文目前使用了 HOG 特征和 LBP 特征进融合，下一步会考虑使用 Haar-like、EOH 等特征进行多特征融合；2)使用多种分块方式来进行分块收缩。本文目前只考虑了将行人样本图像划分成大小 16×16 的分块，这样只能描述行人的局部特征。为了能更好地描述行人的结构特征，下一步准备尝试将分块的大小改为 24×24 或 32×32，从中挑选出分类效果更好的分块进行特征收缩；3)遮挡处理也是我们下一步研究工作的重点。

参 考 文 献

[1] Dalal N, Triggs B. Histograms of Oriented Gradients for Human Detection[C]// Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. San Diego, CA, United states, 2005: 886-893

[2] Zhu Q, Avidan S, Yeh M C, et al. Fast human detection using a cascade of histograms of oriented gradients[C]// Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. New York, NY, United states, 2006: 1491-1498

[3] Mu Ya-dong, Yan Shui-cheng, Liu Yi, et al. Discriminative local binary patterns for human detection in personal album[C]// Proceedings of the 26th IEEE Conference on Computer Vision and Pattern Recognition. Anchorage, AK, United states, 2008

[4] Yao Wen-tao, Deng Zhi-dong. A robust pedestrian detection approach based on shapelet feature and Haar detector ensembles [J]. Tsinghua Science and Technology, 2012, 17(1): 40-50

[5] Levi K, Weiss Y. Learning object detection from a small number of examples: The importance of good features[C]// Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Washington DC, United states, 2004: 1153-1160

[6] Wu Bo, Ram N. Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors[C]// Proceedings of the 10th IEEE International Conference on Computer Vision. Beijing, China, 2005: 90-97

[7] Wu Jian-xin, Rehg James M. CENTRIST: A visual descriptor for scene categorization[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2011, 33(8): 1489-1501

[8] Lowe D G. Distinctive image features from scale-invariant keypoints[J]. International Journal of Computer Vision, 2004, 60(2): 91-110

[9] Lowe D G. Object recognition from local scale-invariant features [C]// Proceedings of the 1999 7th IEEE International Conference on Computer Vision. Kerkyra, Greece, 1999: 1150-1157

[10] Sabzmeydani P, Mori Greg. Detecting pedestrians by learning shapelet features[C]// Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Minneapolis, MN, United states, 2007

[11] Viola P, Jones M. Rapid object detection using a boosted cascade of simple features[C]// Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Kauai, HI, United states, 2001: 1511-1518

[12] Heng C K, Yokomitsu S, Matsumoto Y, et al. Shrink boost for selecting multi-LBP histogram features in object detection[C]// Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Providence, RI, United states, 2012: 3250-3257

(上接第 244 页)

[16] Williams R. An extremely fast ZIV-Lempel data compression algorithm[C]// Data Compression Conference. Snowbird, UT, 1991: 362-371

[17] Gunderson S. Snappy [OL]. http://code.google.com/p/snappy/

[18] Reinhold L M. QuickLZ[OL]. http://www.quicklz.com

[19] Hidayat A. Fastlz[OL]. http://www.fastlz.org

[20] Lenhardt R, Alakuijala J. Gipfeli-high speed compression algorithm[C]// Data Compression Conference. Snowbird, UT, 2012: 109-118

[21] YannCollet. LZ4[OL]. https://code.Google.com/p/lz4/

[22] Sayood K. Introduction to Data Compression (Third Edition) [M]. Singapore: Elsevier, 2009

[23] Ghemawat S, Gobioff H, et al. The google file system [C]// ACM symposium on operating systems principles. New York: USA, 2003: 29-43

[24] Mell P, Grance T. The NIST Definition of Cloud Computing [J]. Communications of the ACM, 2010, 53(6): 50-57

[25] Pereira R, Azambuja M, Breitman K, et al. An architecture for distribute high performance video processing[C]// IEEE International Conference on Cloud Computing. Miami, FL, 2010: 482-489

[26] http://hadoop.apache.org/