doi:10.11887/j.cn.201706009　　http://journal.nudt.edu.cn

分布式计算环境下的栅格数据存储策略*

张剑波,夏灯城,赵加奥,李谢清,崔永键,袁国斌
(中国地质大学 信息工程学院,湖北 武汉 430074)

摘　要:针对传统的栅格数据存储策略不能满足分布式计算环境下粗粒度数据访问需求,应对海量栅格数据计算时效率低下的问题,结合分布式文件系统的存储特点,同时考虑地图代数算子在 Map/Reduce 阶段以栅格瓦片为单位的计算特点,提出一种基于 Hadoop 分布式文件系统的栅格瓦片存储策略。围绕栅格数据瓦片分割、压缩瓦片数据组织与存储、分布式文件输入输出接口改进等方面对该存储策略加以实现,并使用基于该存储策略的地图代数局部算子的分布式计算流程加以验证。理论分析与实验结果表明,该策略能够显著提高分布式计算环境下空间分析算子的运算速度。

关键词:分布式计算;栅格数据;存储策略;地图代数
中图分类号:TP311;P208　　**文献标志码**:A　　**文章编号**:1001-2486(2017)06-051-08

Storage strategy of raster data under the distributed computing environment

ZHANG Jianbo, *XIA Dengcheng*, *ZHAO Jiaao*, *LI Xieqing*, *CUI Yongjian*, *YUAN Guobin*
(Faculty of Information Engineering, China University of Geosciences, Wuhan 430074, China)

Abstract: Traditional storage strategy of raster data cannot meet the demands of coarse-grained data processing under the distributed computing environment and has low efficiency when dealing with calculations for gigantic raster data. A storage strategy of raster tile data was presented on the basis of the storage characteristics of distributed file system. It also took the calculation characteristics of spatial analysis operators of map algebra into consideration, which uses raster tile as processing unit during map and reduce stage. The storage strategy was implemented by the following steps. Firstly raster data were divided into raster tiles. Then these tiles were compressed and organized by a special sequence in order to be transferred to Hadoop distributed file system. Finally input and output file interfaces were re-implemented to meet the data access requirements of map and reduce stage. The strategy was tested and verified by the distributed calculation process of local map algebra operators. Theoretical analysis and experimental results show that this strategy can significantly improve the processing speed of space analysis operators.

Key words: distributed computing; raster data; storage strategy; map algebra

在智慧城市的发展进程中,包含海量非同源高精度遥感数据在内的栅格数据分析和处理能力相对于其数据的获取速度已明显滞后,栅格数据的空间分析高性能计算已成为空间信息科学研究的热点问题[1-2]。近年来,基于栅格数据空间分析的并行加速研究已逐步开展并持续深入,主要呈现四种发展格局:第一种是结合栅格数据分析的计算密集型特点,利用图形处理器(Graphics Processing Unit,GPU)等异构计算设备进行细粒度算法加速[3-6];第二种是利用图形处理器+共享存储并行编程(Open Multi-Processing,OpenMP)+消息传递接口(Message Passing Interface,MPI)混合并行编程模型,实施基于 GPU 集群大规模并行计算[7-10];第三种是利用图形处理器+众核(Many Integrated Core,MIC)+消息传递接口的异构协同计算模型,实施基于众核多线程的大规模数据并行计算[11-14];第四种是利用现有 Hadoop 中的并行编程框架 MapReduce 实现基于计算机集群的分布式并行计算[15-18]。

与栅格数据并行计算模型的多样性研究相比,目前国内外学者针对分布式计算环境下的栅格数据存储模型研究较少。“直接将栅格数据交由分布式文件系统(Hadoop Distributed File System,HDFS)分块划分、实施以像元为单位的空间分析运算”策略[19-20],没有考虑栅格数据“位”分布、“邻”处理的特点,不能完全满足地图代数空间分析算子的分布式计算需求。

栅格数据的存储策略应该是针对其快速计算

* 收稿日期:2016-07-13
基金项目:国家自然科学基金资助项目(41001225,41501584)
作者简介:张剑波(1975—),男,湖北武汉人,副教授,博士,硕士生导师,E-mail:zjb_tigers@126.com

的需求而“定制”的。为此本文设计了一种“HDFS 分块—栅格压缩瓦片”的两级分布式存储模型。它以压缩后的栅格瓦片而非单个像元作为 MapReduce 阶段的最小处理单元,采用“计算近存储器”的数据划分策略,在保留栅格数据空间拓扑关系的基础上,实现了地图代数算子在分布式计算环境下的有效并行加速。

1 HDFS 的栅格瓦片存储策略

1.1 HDFS 的适用性分析

HDFS 具有拍字节(PetaByte, PB)级海量数据管理、基于冗余副本策略的数据容错处理、基于简单一致性模型的流式数据访问等特点。HDFS 适用于栅格数据的分布式存储,主要体现在下面两个方面。

1.1.1 *数据组织*

HDFS 以数据分块(Block)作为最小存储单位,它通过将文件划分为若干个大尺度的数据分块,在最小化硬盘寻址开销的同时,便于数据备份以提高数据容错能力和可用性。类似地,栅格数据的物理存储采用“金字塔层—波段—数据分块”的多级索引机制进行组织,基于这种多级索引结构,在使用栅格数据进行分析时可快速定位到“数据分块”级(即 Tile,栅格瓦片),有效地提高栅格数据存取速度。由此可见,可以利用 HDFS 的大尺度分块一次性存储多个栅格瓦片,进一步缩短空间分析计算过程中对栅格数据分块的访问时间。

1.1.2 *数据访问*

HDFS 采用“一次写入、多次读取”的流式数据访问模式,主要适用于长时间在一个数据集的大部分甚至全部数据上进行分析的应用场景,适用于以栅格数据变换和运算为基础的地图代数空间分析计算。地图代数建模语言主要定义了四种类型的高阶函数(局部、邻域、区域和全局),以局部函数为例,其调用形式如 OutRas = LPos(InRas1, InRas2, …),其计算过程为一次性读入多层栅格数据文件、逐个判断栅格像元值与指定值的关系、进行统计计算、将运算后的数据写入新的栅格数据文件。这样的流式数据处理过程与 HDFS 提供的数据访问方式刚好一致,满足地图代数算子的计算需求。

1.2 栅格数据的瓦片存储策略

HDFS 以输入字节流的顺序将数据读入并划分到多个数据分块的处理方式,可能会将一个栅格瓦片分割到两个不同的数据分块中,从而影响地图代数算子在 MapReduce 阶段处理的速度和精度。同时,不同栅格数据源都采用图像压缩算法(如行程、ZIP 等)对栅格瓦片实施压缩存储以减少计算过程中的数据吞吐量,由此获取的栅格瓦片呈现不定长特点,如直接交由 HDFS 进行划分,可能会出现一个栅格瓦片跨越两个数据分块的情况。

为了解决上述问题,设计的分布式存储策略采用栅格瓦片和 HDFS 数据分块两个层次的划分方式,将栅格数据存储于 HDFS 文件系统中。第一步,在本地以流式切割的方式将栅格数据划分为大小相等的栅格瓦片,并预先实施 ZIP 无损压缩,形成包含元数据信息的一组不定长栅格瓦片集合集,形如 Ras = 〈MetaData, Tile1, Tile2, …〉。第二步,采用自定义数据写入接口方式,将不定长栅格瓦片提交到 HDFS 的数据分块中进行存储,解决了“瓦片存储跨数据分块”的问题。

1.3 存储相关参数的确定

HDFS 以数据分块为单位进行数据处理与分析,默认为 64 MB 或 128 MB,且每个分块默认被存储三份。选取 128 MB 作为数据分块大小(Hadoop-2.6.0 版本的默认分块大小)。而地图代数算子以连续型栅格(浮点型 4 B)和主题型栅格(整型 4 B)两类数据为处理对象,配合前文 128 MB的 HDFS 分块大小,采用 256 KB 作为栅格瓦片大小(即每个瓦片 256 × 256 × 4 B)。这样可以保证每个 HDFS 分块中存储 512 个完整的栅格瓦片,从而避免同一个栅格瓦片跨越两个 HDFS 分块的情况出现。

基于 HDFS 的栅格瓦片数据存储策略的实施过程如图 1 所示。

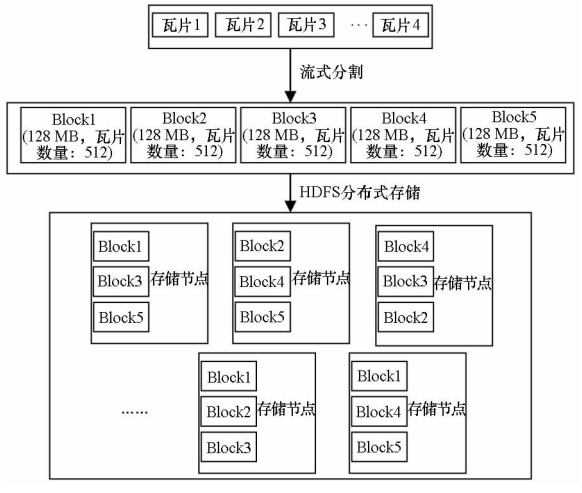


图 1 基于 HDFS 的栅格瓦片数据存储

Fig. 1 Data storage of raster tiles based on HDFS

但是,对栅格数据以 256 × 256 大小实施划分,可能会出现不足一个瓦片大小的情况。常规策略是采用无效值(如 -99 999)进行瓦片填充,导致上传到 HDFS 的实际数据量大于原始栅格数据的大小。为此,采用将栅格瓦片数据先进行压缩、然后存储至 HDFS 中的策略,这样既节省了数据分块冗余存储带来的磁盘空间,又减少了 MapReduce 计算过程中的网络传输开销。

栅格瓦片数据压缩与解压的实现原理描述如图 2 所示。

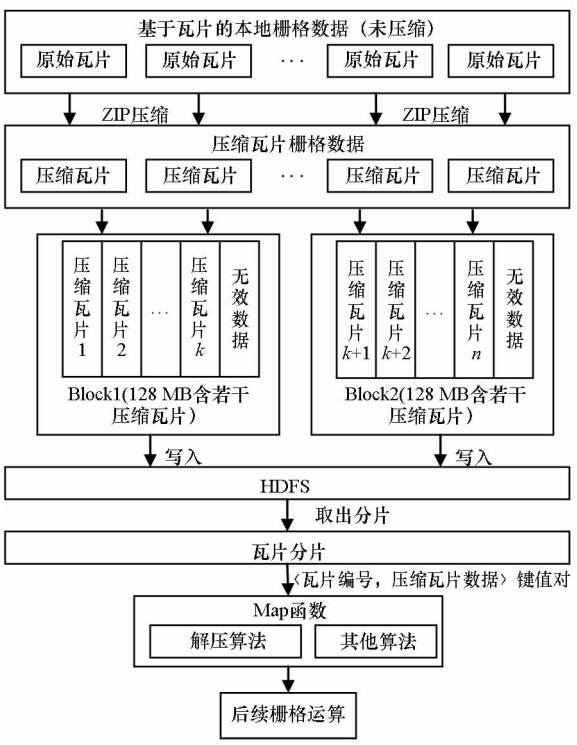


图 2　栅格瓦片数据压缩与解压实现原理

Fig. 2　Realization principle of data compression and decompression of raster tiles

2　存储策略的实现与算子验证

2.1　流程及算法的理论分析

2.1.1　栅格瓦片分割的合理性

空间分块划分是目前栅格数据常用的一种空间索引建立方式,相对基于空间曲线填充(Hilbert 曲线等)的降维、编码和划分方式,它保留了栅格数据行/列各向异性,具有拓扑结构简单、数据调度机制易于实现等优点。

常见的栅格数据空间分块划分可分为条带和矩形两种方式。条带分块方式一般用于栅格数据在行/列方向上分布不均匀的情况(如从卫星上直接获取的条带状 Raw 格式影像)。矩形分块方式(通常为正方形瓦片)具有分块内像素数目相等、分块间空间相关的特点,充分考虑了栅格数据"位""邻""近""势"的特点,更适于栅格数据的空间分析。再结合存储策略,能够保证在分布式计算过程中相邻的像素被划分到同一个 HDFS 分块中,减少节点间的数据传输开销,提高计算效率。

2.1.2　瓦片压缩与解压的合理性

对栅格瓦片实施压缩与解压操作可以减少计算过程中的数据吞吐量,由此带来的计算时间开销分布在两个阶段。一是数据的预处理阶段,该阶段进行瓦片的分割与压缩操作,其计算是在栅格数据提交到 HDFS 之前由客户端完成,不占用分布式计算资源。二是分布式计算阶段,该阶段进行瓦片的解压操作,其计算是在 Map 阶段进行,由此增加的计算时间将被数据压缩减少的磁盘输入输出(Input/Output, I/O)时间所隐藏,体现了分布式环境下以中央处理器(Central Processing Unit, CPU)计算能力降低磁盘读写性能瓶颈的策略(后续实验得以验证)。

2.1.3　空间分析算子的并行性

基于地图代数的空间分析算子可定义为 $OutGrid = Func(\langle Grid_1, Grid_2, \cdots, Grid_n \rangle)$,其中,$\langle Grid_1, Grid_2, \cdots, Grid_n \rangle$ 为只读栅格,OutGrid 为输出栅格,OutGrid 中每个像元的值仅依赖于 $< Grid_1, Grid_2, \cdots, Grid_n >$ 中的值,因而不存在"写后读"或"读后写"的情况。同时,空间分析算子的串行实现是对 OutGrid 中的每个像元进行遍历求解,而且 OutGrid 中的每个像元仅会被计算一次,也不存在"写后写"的情况。根据 Bernstein 条件[21],空间分析算子的多个像元之间的计算是可并行执行的。同时,根据 Amdahl 定律[22],处理器的个数越多、处理器的数据处理能力越强、内/外存支持的 I/O 带宽及重叠特性越完备,空间分析算子的数据处理效率、数据加载和保存效率就越高,可扩展性就越强。因此,依据"数据并行"和"任务并行"的思想,可以将输入/输出栅格分割成若干大小相等的数据瓦片,分别交由分布式环境中不同计算节点实施独立计算,以获取各个数据瓦片的计算结果,从而达到任务的并行。

2.2　栅格瓦片分割

采用如图 3 所示的分割方式对单个栅格数据文件进行瓦片分割,将生成的元数据信息文件(见表 1)上传至 HDFS 保存,栅格瓦片文件则等待后续的压缩处理。

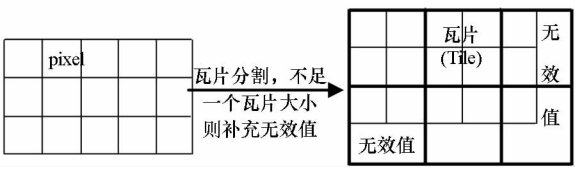


图 3 瓦片分割方式

Fig. 3 Tile partition method

表 1 元数据信息表

Tab. 1 Metadata information

元信息要素	类型	元信息表示意义	大小/B
TileSize	short	瓦片规格	2
ColNum	short	像素列数	2
RowNum	short	像素行数	2
*x*Min	double	*x* 方向像素点的最小值	8
*x*Max	double	*x* 方向像素点的最大值	8
*y*Min	double	*y* 方向像素点的最小值	8
*y*Max	double	*y* 方向像素点的最大值	8
*z*Min	double	*z* 方向像素点的最小值	8
*z*Max	double	*z* 方向像素点的最大值	8
ColTileNum	short	瓦片列数	2
RowTileNum	short	瓦片行数	2
ColAddNum	short	添加的无用像素列数	2
RowAddNum	short	添加的无用像素行数	2

2.3 瓦片压缩流程

2.3.1 压缩瓦片的数据组织

瓦片数据本质是字节流数据，考虑到每个瓦片数据经过 ZIP 压缩后的长度不尽相同，同时压缩瓦片数据在分片后要将瓦片编号和压缩瓦片字节数组分别作为 Key/Value 传递给 Map 函数处理，因此定义了一种适合 Map/Reduce 阶段处理的压缩瓦片数据流格式。压缩瓦片数据组织如图 4所示。

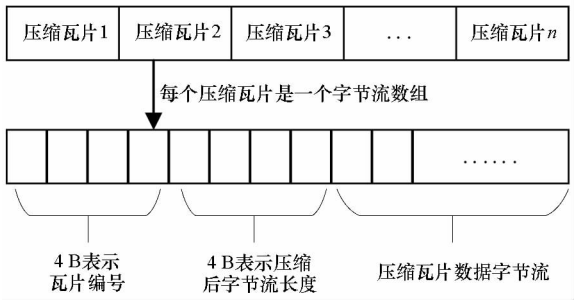


图 4 压缩瓦片数据组织

Fig. 4 Compressed tile data organization

如图 4 所示，数据组织将每个瓦片编号和压缩后的长度（共 8 B），记录在其压缩数据字节流之前，以便后续从分片（Split）中解析并获取瓦片数据时使用。

2.3.2 压缩瓦片的存储

为了避免同一个压缩瓦片数据被分到两个数据分块中而导致无法正确解压的情况，可以对压缩瓦片实施以数据分块大小（默认 128 MB）为单位的分片拼接，以保证每个分片存储整数个压缩瓦片数据。栅格瓦片压缩存储算法的详细步骤见算法 1。

算法 1 栅格瓦片压缩算法

Alg. 1 Raster tile compression algorithm

Input：A：MetaData File
B：TileData File
Output：Compressed TileData File

步骤：

```
1. tileSize← readMetaData(A)
2. ZipFile← newZipFile()
3. zipCount← 0
4. tileListB← getTileList(B,tileSize)
5. foreach B_i ∈ tileListB do
6.    zip B_i← Zip(B_i)
7.    zipB_iLength← getZipB_iLength(zipB_i)
8.    saveZipB_iMetaData(i,zipB_iLength,zipB_i)
9.    zipB_iLength← zipB_iLength + 8
10.   if(zipCount + zipB_iLength < 128MB) then
11.    writeToZipFile(zipB_i, ZipFile)
12.          zipCount + = zipB_iLength
13.          i←i + 1
14.   else
15.          writeNoVaild(ZipFile,128MB)
16.          zipCount← 0
17. output ZipFile and upload ZipFile to HDFS
```

算法 1 中，第 1 ~4 行是算法的预处理阶段，用于信息获取和文件初始化等操作；第 5 行开始对每个栅格瓦片进行压缩处理，使用 ZIP 算法压缩每个瓦片数据，并将瓦片编号、压缩后的字节长度和瓦片数据合并为压缩瓦片信息流，写入 ZipFile 文件中保存；第 17 行将生成的 ZipFile 文件上传到 HDFS 保存。

2.4 输入输出接口改进

Hadoop 在执行一个作业（Job）的时候，会将输入文件划分为若干个分片，然后启动与分片相同个数的 Map Task 程序来分别处理这些

分片。为了使压缩后的栅格瓦片数据集能适用于 MapReduce 计算过程,首先要保证压缩瓦片被 Map 函数正确解析和使用;其次,当瓦片栅格数据完成 Reduce 阶段计算后,还需要将瓦片数据集重组为栅格数据文件格式。因此,需要对现有 MapReduce 输入/输出接口进行改进。

2.4.1　文件划分与输入接口

一是分片函数 InputFormat. getSplit 的实现。每个分片的大小应遵循下面的公式:SplitSize = max{minSize, min{maxSize, BlockSize}}。其中,minSize 表示最小分片大小(默认为 1),maxSize 表示最大分片大小(默认大于或等于数据分块大小 128 MB)。为了避免每个瓦片不被分割到两个不同的分片中,可以使 SplitSize = BlockSize。二是获取键值对函数 RecordReader. nextKeyValue 的实现,将存储多个瓦片的分片解析成为〈tileNo, tileData〉的瓦片键值对形式(其中 tileNo 表示瓦片在栅格数据中的实际位置编号,tileData 表示瓦片的压缩数据流)。

2.4.2　输出接口

一是 Reduce 阶段执行完毕后计算结果输出函数 TileOutputFormat. TileRecordWriter. Write 的实现,将瓦片数据集按编号递增顺序回写到磁盘。二是基于 Hadoop 自带的 TotalOrderPartitioner(全排序分区器),均衡并协同多个 Reduce 任务,保证多个输出文件中瓦片的整体有序性,合并为最终的栅格数据文件。

2.5　算子执行验证

为了验证基于 HDFS 的栅格瓦片存储策略的有效性,选取地图代数局部算子 LocalMax(在多层栅格数据叠加分析中计算对应位置像元的最大值)作为测试对象,通过定义 MapReduce 计算框架下的算子实现流程,说明该存储策略对分布式计算的适用性。LocalMax-MR 算法的详细步骤见算法 2。

算法 2 中,第 1 ~2 行是采用算法 1 对输入栅格数据实施基于 HDFS 的栅格瓦片压缩存储;第 3 ~8行是 Map 处理阶段,对每一个压缩瓦片进行解压缩,并按照〈tileNo, tileData〉键值对形式输出;第 9 ~15 行是 Reduce 阶段,对 tileDataList 中的多块瓦片数据流实施算子操作(如 LocalMax),按瓦片编号顺序输出〈tileNo, resultTileData〉键值对;最后合并多个 resultTileData 得到计算结果文件 C。图 5 给出 LocalMax 算子的分布式计算流程。

算法 2　局部最大值的 MapReduce 算法

Alg. 2　Local max MapReduce algorithm

Input: Raster Data A and B with the same resolution
Output: Raster Data C

步骤:
1. RasterTILE-ZIP (A)
2. RasterTILE-ZIP (B)
   In Map phase:
3. splitsListA← getSplit(A)
4. splitsListB← getSplit(B)
5. foreach split ∈ {splitsListA, splitsListB} do
6. 〈tileNo, zipTile〉← nextKeyValue(split)
7. tileData← unZip(zipTile)
8. output〈tileNo, tileData〉
   In Reduce phase:
9. foreach tileNo ∈ tileNos do
10. tileNo←tileNo
11. tileDataList←tileDataList
12. resultTile←tileDataList[0]
13. foreach i ∈ tileDataList(i > =1) do
14. resultTileData←
    max(resultTile, tileDataList[i])
15. output 〈tileNo, resultTileData〉
16. C ← merge(resultTilesData)

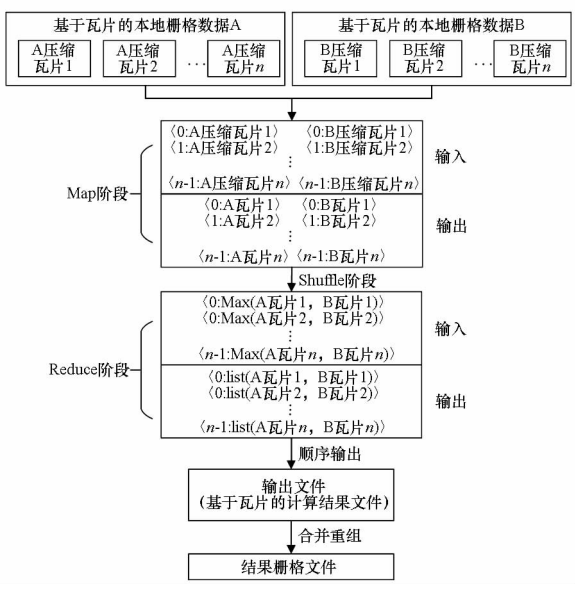


图 5　LocalMax 算子计算流程

Fig. 5　Calculation process of LocalMax

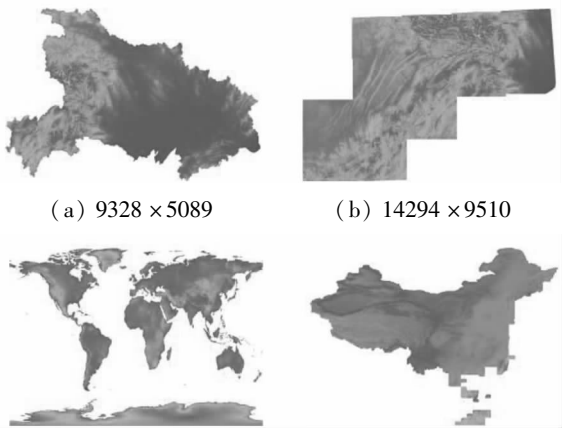
3　实验与分析

3.1　实验环境

实验环境是由 14 个节点构成的 Hadoop 集群。其中 2 台机器作为 NameNode 节点(内存大小 8 GB,

硬盘大小 500 GB),分别作为运行(Active)状态和待命(Standby)状态;12 台机器为 DataNode 及计算节点(内存大小 4 GB,硬盘大小 500 GB)。每台节点上安装的系统为 CentOS6.5(64 位)。Hadoop 版本为 Hadoop-2.6.0,JDK 版本为 JDK1.7。

实验数据来源于全国 1∶250 000 数字高程模型(Digital Elevation Model, DEM)数据库(依据省界范围进行裁切)和全球数字高程模型 DEM 数据 GTOPO 30(采样间隔为 30″),采用四组不同分辨率的数据集:9328 行×5089 列(二进制数据大小约为 181.1 MB)、14 294 行×9 510 列(二进制数据大小约为 518.6 MB)、21 601 行×43 201 列(二进制数据大小约为 3.5 GB)、77 400 行×55 200 列(二进制数据大小约为 15.9 GB),其高程晕渲效果如图 6 所示。



(a) 9328×5089 (b) 14294×9510

(c) 21 601×43 201 (d) 77 400×55 200

图 6 实验数据高程晕渲图

Fig. 6 Elevation shading map of experimental data

3.2 实验结果与分析

3.2.1 实验一

用来测试基于瓦片存储策略的有效性。结合表 2 和图 7 可知,采用瓦片存储策略能够有效提高栅格数据的运算效率。这是由于非瓦片存储策略以栅格数据中的像元为单位进行存储与计算,而瓦片存储策略以栅格瓦片为单位进行数据的存储与计算,更符合分布式计算环境下的粗粒度数据访问需求和运算特点。此外,数据集大小对加速效果也有一定影响。当数据量较小时,计算时间受作业初始化、节点间信息交换等时间开销影响较大,在置信区间内存在波动。随着数据量的增大,瓦片存储策略的优势逐步凸显。

表 2 采用策略和未采用策略运算时间对比

Tab. 2 Comparison of computation time between strategy adopted and non-strategy

数据集	不采用策略运算时间/s	采用策略运算时间/s	加速比
1	191	33	5.8
2	264	47	5.6
3	996	175	5.7
4	4956	413	12

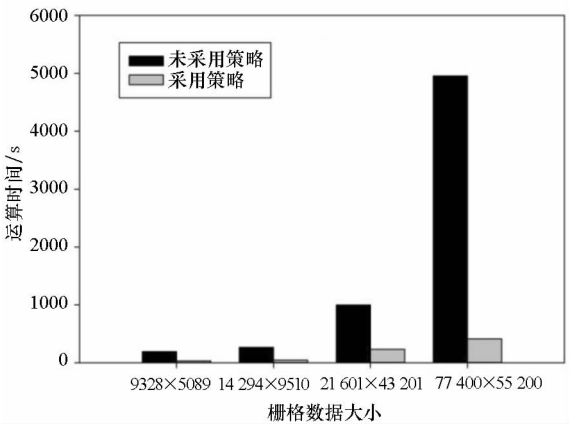


图 7 栅格数据存储的策略执行对比

Fig. 7 Execution comparison of storage strategy of raster data

3.2.2 实验二

用来测试瓦片数据集的压缩效果。结合图 6、图 8 和表 3 可以看出,由于四个数据集的信息熵不同(DEM 数据表征为高程熵),其压缩比存在着较大差异;采用瓦片压缩存储策略能够同时减少栅格数据集在 HDFS 上所需存储空间和在分布式运算时的数据传输时间。虽然瓦片数据压缩和解压操作会带来一定的时间开销,但是实验结果表明:相对于运算时间,数据传输时间更制约整体运算速度。随着栅格数据量的增大,压缩瓦片相对未压缩瓦片的运算效率的提升更加明显。

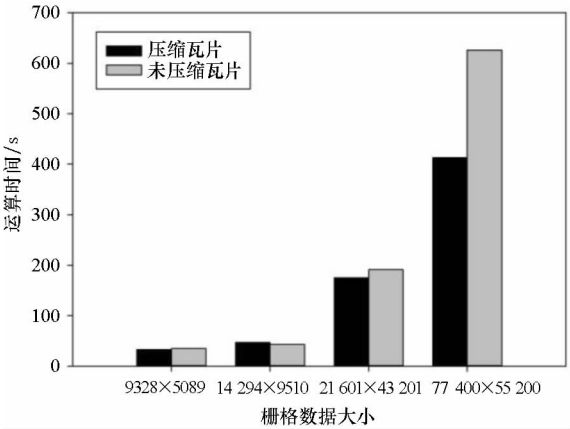


图 8 压缩瓦片与未压缩瓦片数据的执行对比

Fig. 8 Execution comparison between compressed tile and uncompressed tile data

表 3　压缩数据与原始数据存储空间对比

Tab. 3　Comparison of storage space between compressed data and original data

数据集	原始数据大小/MB	压缩后数据大小/MB	压缩比
1	185	24.7	7.5
2	532	185.5	2.9
3	3584	300.7	11.9
4	16 384	1639	10.0

3.2.3　实验三

用来测试计算节点个数对瓦片存储与运算的影响。实验结果如图 9 所示,横轴表示集群中计算节点的个数(不包含主节点),纵轴表示利用分辨率为 77 400 × 55 200 的栅格数据集执行 LocalMax 算子的运算时间。可以看出,随着计算节点个数的增加,在提高任务并行度的同时,还可以使瓦片数据集在集群节点上的分布更加均匀,有利于提高 Map 阶段任务在集群环境中的本地执行效率,从而降低了瓦片数据集的整体运算时间。这也从侧面验证了瓦片分割策略对集群环境扩展的良好适应性。

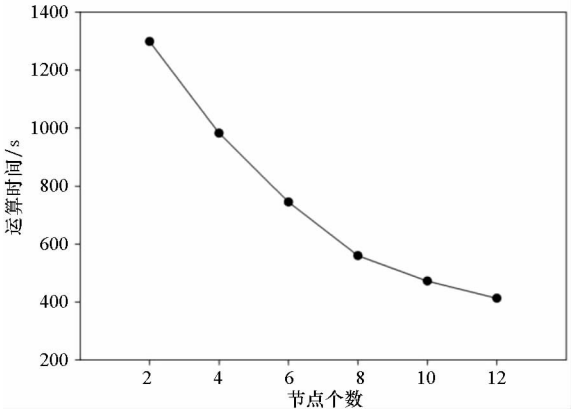


图 9　节点个数对瓦片数据集运算时间的影响

Fig. 9　Influence of number of data node on execution time

3.2.4　实验四

用来测试瓦片存储策略对不同类型算子和应用场景的适用性。实验选取局部(LocalMax)、数学(Add)、全局(Reclassify)和关系(GreaterThan)四类算子,并选取分辨率为 77 400 × 55 200 的栅格数据集进行三种不同尺寸大小(128 × 128, 256 × 256 和 512 × 512)的瓦片切割和运算,实验结果如图 10 所示。实验结果表明,瓦片分割策略对不同类型算子都有很好的适用性。一方面,瓦片尺寸大小直接影响栅格数据的运算速率。瓦片尺寸过大,将导致单次 Map 任务处理数据过多而计算负载增加;瓦片尺寸太小,由瓦片键值对数量增加带来的频繁 I/O 操作,也会增大系统开销从而降低计算速率。综合看来,256 × 256 的瓦片大小更接近 Hadoop 并行运算时的最佳粒度。另一方面,算子的不同计算特点又决定了各自的最佳瓦片分割尺寸。要实现某类算子的最优加速效果,应该结合其算法实现过程来选择合适的瓦片分割尺寸。

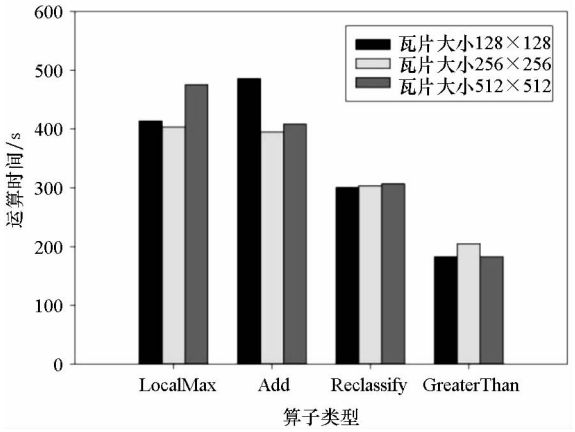


图 10　不同算子类型的瓦片分割尺寸执行对比

Fig. 10　Execution comparison of tiles in different dimension and different algorithm

4　结论

本文针对栅格数据空间分析的分布式计算需求,提出一种基于 HDFS 的栅格瓦片压缩存储策略,最大程度利用分布式计算粗粒度划分和细粒度执行的特点,针对地图代数算子的分布式计算取得了较好的并行加速效果。下一步研究将从适配该存储策略的地图代数并行计算模型和分布式环境下通信优化方法两个方面着手,进一步提高空间分析分布式计算的执行性能。

参考文献(References)

[1]　Guan Q F, Kyriakidis P C, Goodchild M F. A parallel computing approach to fast geostatistical areal interpolation[J]. International Journal of Geographical Information Science, 2011, 25(8): 1241 – 1267.

[2]　Qin C Z, Zhan L J, Zhu A. How to apply the geospatial data abstraction library (GDAL) properly to parallel geospatial raster I/O? [J]. Transactions in GIS, 2014, 18(6): 950 – 957.

[3]　张剑波, 周斯波, 袁国斌, 等. 异构环境下的空间分析并行映射策略[J]. 上海交通大学学报, 2013, 47(1): 70 – 75.
ZHANG Jianbo, ZHOU Sibo, YUAN Guobin, et al. Parallel processing mapping strategy of spatial analysis under the heterogeneous environment[J]. Journal of Shanghai Jiaotong

University, 2013, 47(1): 70 – 75. (in Chinese)

[4] 康俊锋, 杜震洪, 刘仁义, 等. 基于GPU加速的遥感影像金字塔创建算法及其在土地遥感影像管理中的应用[J]. 浙江大学学报(理学版), 2011, 38(6): 695 – 700.
KANG Junfeng, DU Zhenhong, LIU Renyi, et al. Parallel image resample algorithm based on GPU for land remote sensing data management[J]. Journal of Zhejiang University (Science Edition), 2011, 38(6): 695 – 700. (in Chinese)

[5] 肖汉, 张祖勋. 基于GPGPU的并行影像匹配算法[J]. 测绘学报, 2010, 39(1): 46 – 51.
XIAO Han, ZHANG Zuxun. Parallel image matching algorithm based on GPGPU [J]. Acta Geodaetica et Cartographica Sinica, 2010, 39(1): 46 – 51. (in Chinese)

[6] Zhang J T, You S M. High-performance quadtree constructions on large-scale geospatial rasters using GPGPU parallel primitives[J]. International Journal of Geographical Information Science, 2013, 27(11): 2207 – 2226.

[7] Qin C Z, Zhan L J, Zhu A X, et al. A strategy for raster-based geocomputation under different parallel computing platforms [J]. International Journal of Geographical Information Science, 2014, 28(11): 2127 – 2144.

[8] 赫高进, 熊伟, 陈荦, 等. 基于MPI的大规模遥感影像金字塔并行构建方法[J]. 地球信息科学学报, 2015, 17(5): 515 – 522.
HE Gaojin, XIONG Wei, CHEN Luo, et al. An MPI-based parallel pyramid building algorithm for large-scale RS image[J]. Journal of Geo-Information Science, 2015, 17(5): 515 – 522. (in Chinese)

[9] 申焕, 石晓春, 邱宏华. 基于MPI的海量遥感影像并行处理技术探析[J]. 全球定位系统, 2012, 37(6): 73 – 76.
SHEN Huan, SHI Xiaochun, QIU Honghua. Study on parallel processing technology of massive remote sensing image based on MPI[J]. GNSS World of China, 2012, 37(6): 73 – 76. (in Chinese)

[10] Didelot S, Carribault P, Pérache M, et al. Improving MPI communication overlap with collaborative polling [J]. Computing, 2014, 96(4): 263 – 278.

[11] Lai C G, Huang M Q, Shi X, et al. Accelerating geospatial applications on hybrid architectures [C]//Proceedings of IEEE 10th International Conference on High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing, 2013: 1545 – 1552.

[12] Shi X, Lai C G, Huang M Q, et al. Geocomputation over the emerging heterogeneous computing infrastructure [J]. Transactions in GIS, 2014, 18(S1): 3 – 24.

[13] Shi X, Ye F. Kriging interpolation over heterogeneous computer architectures and systems[J]. GIScience & Remote Sensing, 2013, 50(2): 196 – 211.

[14] Bernabé S, Sánchez S, Plaza A, et al. Hyperspectral unmixing on GPUs and multi-core processors: a comparison[J]. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2013, 6(3): 1386 – 1398.

[15] 刘磊, 尹芳, 冯敏, 等. 基于开源Hadoop的栅格数据分布式处理[J]. 华中科技大学学报(自然科学版), 2013, 41(7): 103 – 108.
LIU Lei, YIN Fang, FENG Min, et al. Distributed computation of raster data using open source Hadoop[J]. Journal of Huazhong University of Science and Technology (Natural Science Edition), 2013, 41(7): 103 – 108. (in Chinese)

[16] 刘义, 陈荦, 景宁, 等. 利用MapReduce进行批量遥感影像瓦片金字塔构建[J]. 武汉大学学报(信息科学版), 2013, 38(3): 278 – 282.
LIU Yi, CHEN Luo, JING Ning, et al. Parallel batch-building remote sensing images tile pyramid with MapReduce [J]. Geomatics and Information Science of Wuhan University, 2013, 38(3): 278 – 282. (in Chinese)

[17] Li J Y, Meng L K, Wang F Z, et al. A map-reduce-enabled SOLAP cube for large-scale remotely sensed data aggregation[J]. Computers & Geosciences, 2014, 70: 110 – 119.

[18] 鲁伟明, 杜晨阳, 魏宝刚, 等. 基于MapReduce的分布式近邻传播聚类算法[J]. 计算机研究与发展, 2012, 49(8): 1762 – 1772.
LU Weiming, DU Chenyang, WEI Baogang, et al. Distributed affinity propagation clustering based on MapReduce [J]. Journal of Computer Research and Development, 2012, 49(8): 1762 – 1772. (in Chinese)

[19] 蒋波涛, 王艳东. 基于MapReduce的地图代数并行计算方法[J]. 测绘地理信息, 2014, 39(3): 50 – 55.
JIANG Botao, WANG Yandong. Map algebra parallel calculation method based on MapReduce [J]. Journal of Geomatics, 2014, 39(3): 50 – 55. (in Chinese)

[20] Zhang G Q, Xie C J, Shi L, et al. A tile-based scalable raster data management system based on HDFS [C]// Proceedings of the 20th International Conference on Geoinformatics, 2012: 1 – 4.

[21] Bernstein A J. Analysis of programs for parallel processing[J]. IEEE Transactions on Electronic Computers, 1966, EC-15(5): 757 – 763.

[22] Amdahl G M. Validity of the single processor approach to achieving large scale computing capabilities [C]// Proceeding of AFIPS67, 1967: 483 – 485.