2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

A parallel implementation of a XDraw viewshed algorithm with Spark

1st Jianbo Zhang
Faculty of Information Engineering
China University of Geosciences
Wuhan, China
zhangjb@cug.edu.cn

2nd Caikun Chen
Faculty of Information Engineering
China University of Geosciences
Wuhan, China
cck0908@126.com

3rd Tingnan Liang
Faculty of Information Engineering
China University of Geosciences
Wuhan, China
liangtingnan@126.com

4th Hao Xia
Faculty of Information Engineering
China University of Geosciences
Wuhan, China
cugxiahao@126.com

5th Simin Zhou
Faculty of Information Engineering
China University of Geosciences
Wuhan, China
zsmas10@126.com

***Abstract*—Viewshed analysis is an indispensable part of digital terrain analysis and is widely used in many application domains. High-resolution raster DEM data bring significant computational challenges to the existing viewshed analysis algorithms, which are computationally intensive and require a large memory space and massive computing power. The viewshed analysis process can be accelerated through the use of Apache Spark. In this article, we present both a tile-based raster data storing strategy and an equal-volume computing strategy for distributed viewshed computation using Spark. The parallel implementation of the XDraw algorithm mainly consists of three parts: (1) partitioning a raster DEM file into square tile sets and reorganizing these tile sets to prevent tile overlap across data divisions of HDFS, (2) subdividing the DEM into multiple equal-volume data sectors according to the viewpoint position, and (3) retrieving the corresponding tile sets of each sector to perform the XDraw algorithm independently and efficiently. Experiments with real-world datasets show that the two proposed strategies can achieve higher speed-up and efficiency for XDraw viewshed analysis as the raster DEM data volume is dramatically increased.**

***Index Terms*—viewshed analysis, XDraw, Spark, parallel computing**

I. Introduction

Terrain viewshed analysis is an important part of digital terrain analysis, which is focused on the terrain range that can be seen from the viewpoint position in the study area, or the visibility between viewpoint and target [1]. It is widely used in military, communication [2], real estate, archaeology [3], landscape design and some other fields.

With the development of survey and mapping technologies, the volume of digital terrain data has reached to an unprecedented scale. The computational load of the viewshed analysis increases exponentially as a result. Mature commercial Geographic Information System software (such as ArcGIS) has been unable to process such large volume of data. It is urgent to improve the computational efficiency of the viewshed algorithm through the high-performance computing (HPC). As a frontier technology in the era of big data, the high-performance geocomputation could schedule hardware and software resources efficiently according to the different characteristics of geographic computing tasks. It could satisfy the computationally intensive need of multi-viewpoint visibility judgment in the process of viewshed analysis.

A. Viewshed analysis

When calculating the viewshed of a target within a large observation area, viewshed analysis utilizing the Line of Sight (LoS) algorithm will cause a lot of repeated calculations if each point calculates its own viewshed independently. Whether to reuse the intermediate calculation result can be balanced between the calculation accuracy and the efficiency. Therefore, the existing viewshed algorithms can be divided into two categories: non-reusable algorithm and reusable algorithm.

(1) The non-reusable algorithm is relatively accurate but time-consuming. The time complexity of this algorithm is O(n^3), where n is the number of topographic points of the raster DEM. A typical example is the R3 algorithm [4]. In order to improve the computational efficiency of the R3 algorithm, the reference surfaces algorithm [5] and its improved version [6] were proposed. But there is still a problem of computing efficiency for large amounts of data. The research on the parallelization of R3 algorithm is focused on how to exploit high parallelism of graphic processing units (GPUs). By designing a proper spatial domain decomposition strategy to achieve efficient Input/Output (I/O) management [1], the modified R3 algorithm implemented on the GPU can also achieve good performance improvement [7].

(2) The reusable algorithm reuses the intermediate calculation result to save the total calculation time at the expense of accuracy. According to the different reusable ways, this category can be subdivided into the reusable approximation point method (the R2 algorithm) and the reusable point-by-point outward method (the XDraw algorithm). They are both run on O(n^2) time.

The R2 algorithm reduces the complexity by calculating the intersection of the sight line and the DEM grid boundary. Some research effort have been directed toward the design and parallel implementation of the R2 algorithm via GPU. The Compute Unified Device Architecture (CUDA) has been proved to be more efficient than CPU in the parallel implementation of the R2 algorithm [8]. Since the R2 algorithm has been criticized for not being input/output (I/O) efficient [9], other efforts focus on solving I/O efficiency problems to obtain the best computing time [10]. In addition, there are other studies contributing to improve the accuracy of the original R2 algorithm [11].

B. XDraw

The XDraw algorithm divides the raster DEM data according to the eight clockwise directions of North, North-East,East, South-East, South, South-West,West and North-West. Instead of calculating a LoS to any given point on the terrain and then progressing to that points neighbor, the XDraw algorithm assumes a regular growth pattern emanating from the observers position on the DEM in which LoS information is gathered [12]. Much research efforts on the parallel acceleration of the XDraw algorithm focus on using the domain decomposition strategy to reduce I/O costs for calculating the viewshed on large data sets [13]–[15], improving the parallel efficiency and the computational accuracy of the algorithm with GPU [12], and optimizing the algorithm to improve the computational efficiency for massive DEM data [16].

General-purpose computation on graphics processing units (GPGPUs) has become the mainstream for speeding up the viewshed analysis algorithm. It needs to be configured with the expensive high-performance workstations equipped with many-core GPUs. By comparison, the distributed framework, Apache Hadoop, is a more economical computing model. Apache Spark is a fast and general-purpose cluster computing framework for large-scale data processing. The combination of geocomputation and the distributed computing framework could achieve high performance for digital terrain analysis.

In addition, the R3 algorithm is slow but highly accurate, the R2 algorithm is faster but less accurate [4]. The advantage of the XDraw algorithm is using the visibility between points for recurrence and diffusion operations, which can reduce the calculation time and gain better calculation accuracy. The spatial independency of the XDraw algorithm makes it possible to using Spark to optimize the algorithm to attain a high level of parallelization.

In this article, a parallel computing approach for the XDraw algorithm using the distributed computing framework (Spark) is presented. Firstly, we propose a tile-based data storage strategy. The strategy solves the raster data storage problem of the Hadoop Distributed File System (HDFS). Then, we design an equal-volume computing strategy for the distributed framework (Spark) to achieve parallel acceleration for the XDraw algorithm. The implementation of the XDraw algorithm includes the following three steps: (1) partitioning a raster DEM file into square tile sets, and reorganizing these tile sets, which can prevent data overlap between tiles. (2) subdividing the DEM into multiple equal-volume data sectors according to the existing viewpoint position. (3) retrieving the corresponding tile sets of each sector and implementing the Spark-based XDraw algorithm based on these tile sets.

II. XDraw Sequential Algorithm

The general idea of XDraw viewshed algorithm is to use the visibility of adjacent points for the recurrence and diffusion operations. Given a viewpoint V in the raster DEM showed in Fig.1, the algorithm takes it as a center, generates an initial ring by diffusing outward in the eight directions step by step, and calculates the visibility of the point on the ring in terms of the visibility of its adjacent points. This computational process continues until the boundary of the DEM is reached. The visibility calculation method of the points on the ring is described as follows:

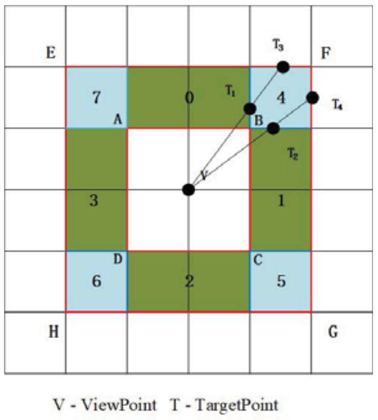

Fig. 1. The principle of the XDraw algorithm

Firstly, the initial ring R_0 (rectangle ABCD) is diffused outward to get a new ring R_1 in the eight directions (North, East, South, West, North-East, North-West, South-East and South-West in turn) with each step of a grid cell. Then the new ring R_1 is divided into eight 45° octants to form rectangles 0-7. The visibility of the points lying in the rectangles is calculated by using the elevation differences between the points on the rectangle ABCD and the viewpoint V. The computing order of the rectangles is 0, 1, 2, 3, 4, 5, 6 and 7. By repeating the steps above, the visibility of the points lying in the R$_{i+1}$ can be deduced from the intermediate calculation results of the Ri and the viewpoint V, and so on. The algorithm ends when the boundary of the DEM is reached. More details of the XDraw algorithm are described as follows:

In the above steps, Steps 1-2 generate an initial ring according to the viewpoint. Steps 3-16 perform the visibility calculation from the inner ring to the outer ring iteratively until the boundary of the DEM is reached. In the output raster data, the cell value 1 represents visible, value -1 represents invisible, and value 0 represents an invalid value.

Algorithm 1 XDraw algorithm

```
Input: the raster DEM $A$, the viewpoint $V$
Output: the binary raster data $B$, where the cell value is 1 denotes
    it is visible
 1: get viewpoint information
 2: get initial ring based on the viewpoint
 3: while $the\ ring\ is\ in\ the\ DEM$ do
 4:     $ring \leftarrow ring + 1$
 5:     determine the order of the rectangles between the two rings
 6:     for each $R_{Ti} \in rectangles(RT)$ do
 7:         calculate whether cells in $R_{Ti}$ are visible by Line of
            visibility
 8:         if $the\ cell\ is\ visible$ then
 9:             $value \leftarrow 1$
10:         else if $the\ cell\ is\ not\ visible$ then
11:             $value \leftarrow -1$
12:         else
13:             $value \leftarrow 0$
14:         end if
15:     end for
16: end while
17: $Output\ the\ binary\ raster\ data$
```

From the implementation of XDraw serial algorithm, we can see that:

(1) The XDraw sequential algorithm could be accelerated by means of the power of parallel computing. The eight sub-rectangles formed in each diffusion process do not overlap with each other, and their executions of the viewshed algorithm are independent. Moreover, there is no data communication and calculation results dependence among the executions.

(2) The original XDraw algorithm adopts a 45° equiangular data partitioning strategy, which could cause load imbalance during the distributed computing process. The location of the viewpoint may cause huge differences of the data volume in the eight sub-rectangles, which could result in a distinctly different amount of data allocated to each computing node in the cluster environment. When the viewpoint lies in the middle area of the DEM, the data amount of each sub-rectangle is approximately the same. However, when the viewpoint position is not in the central area of the DEM (such as on the boundary), the data amount of each sub-rectangle varies greatly. It will lead to the load imbalance of the distributed computing and reduce the efficiency of parallel computing as a consequence. Therefore, this article focuses on designing an equal-volume computing strategy to improve the parallel computing performance of the Spark-based XDraw algorithm effectively.

III. Design of the Proposed Strategies

The distributed XDraw algorithm uses a tile-based raster data storing strategy and an equal-volume computing strategy on Spark for a raster DEM data. Firstly, the DEM data is partitioned into square tile sets before submitted to HDFS. Secondly, the DEM data is logically subdivided into multiple equal-volume sectors according to the position of the viewpoint. Then, the Spark-based XDraw algorithm is performed on each sector. Finally, the viewshed results of all sectors are merged and saved persistently. The flowchart of the XDraw algorithm on Spark is illustrated as Fig.2.

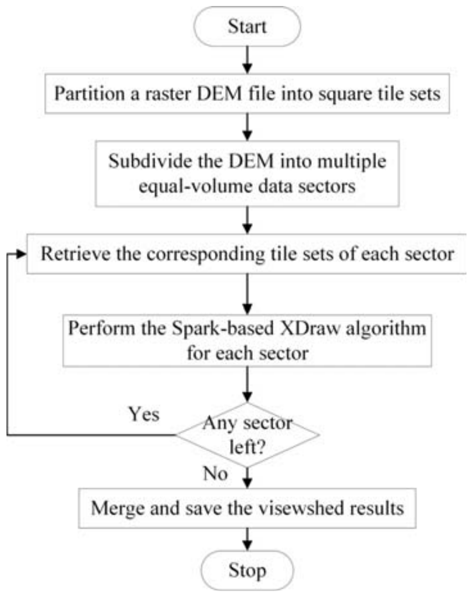
Start
Partition a raster DEM file into square tile sets
Subdivide the DEM into multiple equal-volume data sectors
Retrieve the corresponding tile sets of each sector
Perform the Spark-based XDraw algorithm for each sector
Any sector left?
Yes
No
Merge and save the visewshed results
Stop

Fig. 2. The flowchart of the XDraw algorithm on Spark

A. Tile-based raster data storing strategy

The reason for using HDFS to store raster data is that the data organization of HDFS is similar to that of raster data.HDFS adapts a data block as a minimum storage unit. Files are split into a series of fixed size data blocks (the default block size is 64 MB) distributed over a cluster of data nodes [17], which take care of replication for fault tolerance and can minimize addressing overhead. Similarly, a pyramid-band-tile hierarchical structure is often used for raster data organization and can help to locate and access data tiles quickly while performing raster-related analysis [18]. The size of a tile is typically smaller than that of a data block. Consequently, multiple raster tiles can be re-organized in the form of byte stream to fill a 64 MB division, which can further reduce the access time to raster tiles during spatial analysis calculation.

However, there is an inevitable problem in using HDFS to store raster data. An HDFS file is divided into 64 MB partitions. This division can make the calculation process of the program as close as possible to the operational data. It helps minimize network congestion and increase overall system throughput [19]. If raster data are submitted to HDFS without any treatment, a raster tile may lie across two divisions on different data nodes of the cluster. Therefore, a proper storing strategy for raster data on HDFS should be carefully considered.

To address this problem, a storing strategy of HDFS divisions was designed. The main idea is to adopt a raster tile (as opposed to a single pixel) as a minimum process unit during the distributed computing stage. The storing strategy was designed to work in two steps:(1) partition the raster data into tile sets with equal rows and columns; and (2) improve the data access interface of HDFS to support the user-defined binary data format, and then submit those tile sets to data divisions of HDFS to persistent storage.

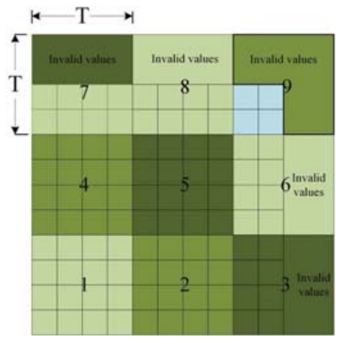

Fig. 3. The tile decomposition strategy

In step (1), how to determine the size of a raster tile is essential to solve the problem of across-HDFS division. The tile decomposition strategy is illustrated as Fig.3. The raster data is divided into multiple square tiles with a specified tile size T.

$$T = 2^n, 1 \leq n \leq \log_2 min(Row, Col) \quad (1)$$

Where Row and Col denote the row and column numbers of the raster data. If the row or column number of a tile is less than T, the blank area of the tile will be filled with an invalid value (such as -99999) to ensure all tiles have the same size. The tile size should be determined dynamically according to the volume of raster data. In this article, 256 was selected as the size of a raster tile in order to ensure each HDFS division containing 512 whole raster tiles. The exceptional situation of one tile across two divisions is therefore avoided.

B. Equal-volume computing strategy on Spark

The original 45° equiangular data partitioning strategy of the XDraw algorithm could cause load imbalance during the distributed computing process when the viewpoint lies on the boundary of the DEM. Therefore, an equal-volume computing strategy was proposed to address this problem. The main idea is to adopt an equal-volume condition as a criterion for logical partitioning of the DEM.

A raster DEM data with M rows and N columns can be regarded as a rectangle whose height is M and width is N, which could be subdivided into multiple equal-volume sectors according to the position of the viewpoint and the pre-set number of sectors. The crucial step of the equal-volume computing strategy is to determine the starting point and ending point of each sector. Also, the lower left corner point of the DEM is regarded as the reference coordinate origin (0,0). The flowchart of the equal-volume computing strategy is illustrated as Fig.4.

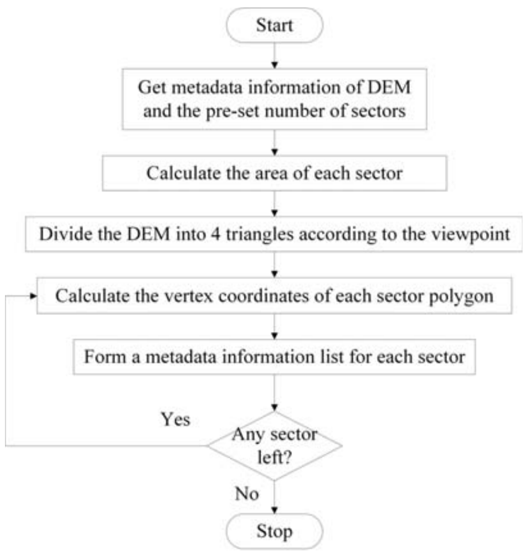
Start
Get metadata information of DEM and the pre-set number of sectors
Calculate the area of each sector
Divide the DEM into 4 triangles according to the viewpoint
Calculate the vertex coordinates of each sector polygon
Form a metadata information list for each sector
Yes
Any sector left?
No
Stop

Fig. 4. The flowchart of the equal-volume computing strategy

The implementation of this strategy includes the following four steps:

(1) Calculate the area of each sector according to metadata information of the DEM and the pre-set number of sectors.

$$S = (Row * Col)/N \quad (2)$$

Where Row and Col denote the row and column numbers of the DEM separately, and N denotes the number of sectors.

(2) Divide the DEM into 4 triangles according to the viewpoint and the minimum bounding rectangle of the DEM.

(3) Calculate the vertex coordinates of each sector polygon. The calculation starts with line VA and is carried out anti-clockwise. The starting point of the next sector polygon is the ending point of the previous sector polygon, and the ending point of the last sector polygon is the reference coordinate origin A.

Specifically, the 4 basic triangles (triangle VAB, VBC, VCD and VDA composed of dotted lines showed in Fig.5) generated by Step(2) are the foundation of this calculation process. The unknown vertices of each sector polygon are determined by comparing the area of the three basic triangles (S_{VAB}, S_{VBC} and S_{VCD}) with the area of the sector S. According to the location of the sector, the calculation process can be summarized in TABLE I.

Where N_i denotes the number of each sector ranging from 0 to the number of sectors minus one.

(4) Form a metadata information list for each sector. The minimum bounding rectangle (MBR) of each sector can be calculated according to the viewpoint coordinate and the vertex coordinates of each sector polygon. Take as an example, the minimum bounding rectangle AEGH of the sector VAE is obtained by comparing the minimum and maximum values of its three vertex coordinates. By repeating the steps above,

TABLE I
THE CALCULATION PROCESS OF THE LOCATION OF EACH SECTOR

Vertices of the sector	Vertex coordinate X	Vertex coordinate Y
both on AB	$X = N_i * S * 2/Y$	$Y = 0$
one on AB and one on BC	$X = Col - 1$	$Y = (N_i * S - S_{VAB}) * 2/X$
both on BC	$X = Col - 1$	$Y = (N_i * S - S_{VAB}) * 2/(Col - 1 - X)$
one on BC and one on CD	$X = Col - 1 - ((N_i * S - S_{VAB} - S_{VBC}) * 2/(Row - 1 - Y))$	$Y = Row - 1$
both on CD	$X = Col - 1 - ((N_i * S - S_{VAB} - S_{VBC}) * 2/(Row - 1 - Y))$	$Y = Row - 1$
one on CD and one on DA	$X = 0$	$Y = Row - 1 - ((N_i * S - S_{VAB} - S_{VBC} - S_{VCD}) * 2/X)$
both on DA	$X = 0$	$Y = Row - 1 - ((N_i * S - S_{VAB} - S_{VBC} - S_{VCD}) * 2/X)$

TABLE II
THE METADATA INFORMATION LIST OF THE SECTOR

Metadata Item	Numeric Type	Meaning	Bytes
SectorNo	Short	Sector number	2
StartPoint	Point(X,Y)	The starting point of the sector	8
EndPoint	Point(X,Y)	The ending point of the sector	8
RectangleStart	Point(X,Y)	The left lower corner of the sectors MBR	8
RectangleEnd	Point(X,Y)	The upper right corner of the sectors MBR	8

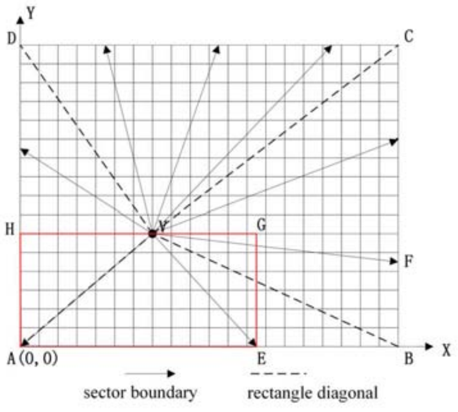

Fig. 5. The equal-volume partitioning strategy

a metadata information list for each sector is formed in terms of the following format described in TABLE II.

IV. Distributed algorithm implementation

A. Partition algorithm of raster tiles on HDFS

The tile partitioning algorithm includes the following three steps:

(1) Obtain the metadata information of the given raster file, and calculate the rows and columns of tiles via a predefined tile size ($TileSize = 256$)

$$TileRowNum = \begin{cases} RowNum/TileSize + 1 \\ \quad if(RowNum \ mod \ TileSize \neq 0) \\ RowNum/TileSize \\ \quad if(RowNum \ mod \ TileSize = 0) \end{cases} \tag{3}$$

$$TileColNum = \begin{cases} ColNum/TileSize + 1 \\ \quad if(ColNum \ mod \ TileSize \neq 0) \\ ColNum/TileSize \\ \quad if(ColNum \ mod \ TileSize = 0) \end{cases} \tag{4}$$

Then, form a metadata file according to this information. TABLE III lists the information of the metadata file.

(2) Partition a raster file into tile sets according to a tile with 256 rows and 256 columns. Note that the tile size (256×256) gives rise to the condition of less than one tile during the execution of the partition algorithm. It could be resolved by filling the blank with an invalid value. The procedure is as follows:

1) Create a tile dataset file, and partition the raster file in terms of the method shown in Fig.6.

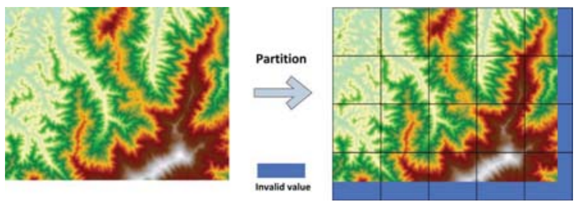

Fig. 6. Partitioning method for geo-raster

2) Calculate the rows and columns of invalid values added.

$$AddRowNum = \begin{cases} TileRowNum * TileSize - RowNum \\ \quad if(RowNum \ mod \ TileSize \neq 0) \\ 0 \\ \quad if(RowNum \ mod \ TileSize = 0) \end{cases} \tag{5}$$

TABLE III
METADATA INFORMATION

Metadata Item	Numeric Type	Meaning	Bytes
TileSize	short	Byte size of a tile	2
ColNum	Integer	Columns of the raster	4
RowNum	Integer	Rows of the raster	4
xMin	double	Minimal x of the raster	8
xMax	double	Maximal x of the raster	8
yMin	double	Minimal y of the raster	8
yMax	double	Maximal y of the raster	8
zMin	double	Minimal z of the raster	8
zMax	double	Maximal z of the raster	8
TileColNum	Integer	Columns of tiles	4
TileRowNum	Integer	Rows of tiles	4
AddColNum	Integer	Columns of invalid-value added	4
AddRowNum	Integer	Rows of invalid-value added	4

$$AddColNum = \begin{cases} TileColNum * TileSize - ColNum \\ \quad if(ColNum \ mod \ TileSize \neq 0) \\ 0 \\ \quad if(ColNum \ mod \ TileSize = 0) \end{cases} \tag{6}$$

3) For each tile, perform the following steps:
   a) If the column number of a tile is less than TileSize, the blank at the end of each row is filled with an invalid value set with the number of AddColNum to form a whole tile.
   b) If the row number of a tile is less than TileSize, the blank at the end of each column is filled with an invalid value set with the number of AddRowNum to form a whole tile.
   c) Other complete tiles are derived from the raster file directly.

(3) Upload these two files to HDFS. The metadata file is stored at the master node, while the tile dataset file is distributed throughout all data nodes.

B. Equal-volume computing algorithm on Spark

In order to reduce the data transferring overhead during the execution of the Spark task, the DEM is divided into multiple equal-volume sectors using the proposed strategy described in Section III. More details of the algorithm are described as Algorithm 2.

In the above steps, Steps 1-3 carry out the equal-volume division on the DEM to obtain multiple sectors and their metadata information list. Steps 4-16, a series of RDD operations are performed on each sector to get its RDD $\langle index, value\rangle$ prepared for the following viewshed algorithm, where index denotes the global index of each grid cell and value denotes the elevation of the cell. Specially, the $map(tileRDD)$ in Step 8 represents the calculation procedure of the global index of each grid cell.

C. Implementation of XDraw algorithm based on Spark

Each sector performs the Spark-based XDraw algorithm. The viewshed results of all sectors are merged to form the visibility raster data saved in HDFS. More details of the algorithm are described as Algorithm 3.

Algorithm 2 Equal-volume computing algorithm on Spark

```
Input: the tile sets $T$, the viewpoint $V$
Output: sector RDD $\langle index, value\rangle$
 1: get metadata information of DEM and the pre-set number of
    sectors
 2: get sectors using the strategy in Section 3.2
 3: store the metadata information of all sectors
 4: for each $S_i \in sectors(S)$ do
 5:     retrieve tiles by MBR of the $S_i$
 6:     for each $T_i \in tiles(T)$ do
 7:         $tile\ RDD\langle index, value\rangle \leftarrow binary\ Records(tile)$
 8:         $real\ RDD\langle index, value\rangle \leftarrow map(tile\ RDD)$
 9:     end for
10:     $rt\ RDD\langle index, value\rangle \leftarrow union(real\ RDD)$
11:     for each $cell\langle index, value\rangle \in rt\ RDD$ do
12:         if $the\ cell\ is\ not\ in\ the\ S_i$ then
13:             $index \leftarrow -1, value \leftarrow invalid\ value$
14:         end if
15:     end for
16:     $sector\ RDD\langle index, value\rangle \leftarrow rtRDD.filter(index > 0)$
17: end for
18: $Output\ the\ sector\ RDD\langle index, value\rangle$
```

In the above steps, Step 1-22 carry out the XDraw viewshed algorithm on the RDD of each sector. Step 23-24 merge and transform the result RDDs.

V. COMPUTATIONAL EXPERIMENTS

A. Datasets

The performance and accuracy of the two proposed strategies in this article was evaluated using the digital elevation model(DEM) of Australia(https://data.gov.au/data/dataset/da926e47-1cd8-4dc9-b859-cbc18c29d858). To determine the impact of varying the data volume on the parallel performance, four elevation datasets of different size were used. Meanwhile, to determine the variation of computation time over different topographic features, three different kinds of viewpoints were selected for each dataset. They included a pit, a peak and a point in flat areas. All dataset files listed in TABLE IV were

TABLE IV
AUSTRALIA ELEVATION DATASETS

Dataset	Size(GB)	Grid-cell size	Columns	Rows	Grid cells
Grid1	0.67	10”×10”	14760	12240	180662400
Grid2	1.05	8”×8”	18450	15300	282285000
Grid3	2.69	5”×5”	29520	24481	722679120
Grid4	7.47	3”×3”	49200	40801	2007409200

Algorithm 3 XDraw based on Spark

```
Input: sector $RDD$, the viewpoint $V$
Output: the binary raster data $B$, where the cell value is 1 denotes
    it is visible
1: for each $S_i$ $RDD \in sector\ RDD(S)$ do
2:   $hashmap\ RDD\langle 1, hashmap\langle index, value\rangle\rangle \leftarrow aggregateByKey\ (S_i\ RDD)$
3:   calculate reference coordinates of the viewpoint $V$
4:   get initial ring based on the viewpoint
5:   while $the\ ring\ is\ in\ the\ DEM$ do
6:     $ring \leftarrow ring + 1$
7:     determine the order of the rectangles between the two rings
8:     for each $R_{Ti} \in rectangles(RT)$ do
9:       for each $cell \in R_{Ti}$ do
10:        calculate the elevation value of the cell
11:      end for
12:      calculate whether cells in $R_{Ti}$ are visible by the XDraw
         algorithm
13:      if $the\ cell\ is\ visible$ then
14:        $value \leftarrow 1$
15:      else if $the\ cell\ is\ not\ visible$ then
16:        $value \leftarrow -1$
17:      else
18:        $value \leftarrow 0$
19:      end if
20:    end for
21:  end while
22: end for
23: $sectors\ RDD\langle 1, \langle index, value\rangle\rangle \leftarrow union(hashmap\ RDD)$
24: $result\ RDD\langle index, value\rangle \leftarrow reduceBykey(sectors\ RDD)$
25: $Output\ the\ binary\ raster\ data$
```

saved in extended GRD format (the Surfer grid file format of GoldenSoft).

The elevation map of Australia is shown in Fig.7.

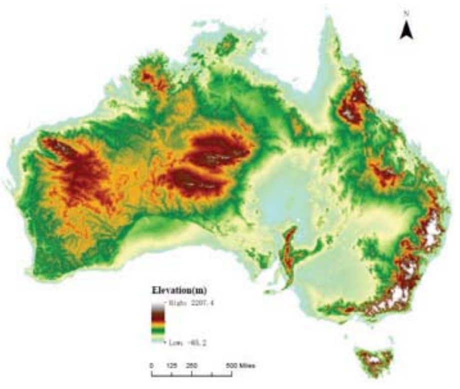

Fig. 7. Elevation map of Australia

B. Hardware Environment

A computer cluster with 2 name nodes and 8 data nodes was used as the hardware platform. These nodes are linked by 100 Mbps fast ethernet. There are 57 cores in all in the compute nodes of this cluster. The CentOS 6.5 operating system, Hadoop 2.6.0 and JDK 1.7 are used for each computer. Meanwhile, a workstation with ArcGIS was used to execute the experimental algorithm to compare pre-processing and calculation time with those of the cluster.

C. Experimental Design

Three experiments were conducted to test the parallel performance of the two proposed strategies. Note that each experiment was repeated five times to obtain an average computing time. The computing time here refers to the execution time for running spark tasks, and the data pre-processing time was also recorded to be compared with it.

(1) To investigate the effectiveness of the proposed strategies on parallel performance, the viewshed analyst tool of ArcGIS on the workstation and the XDraw algorithm implemented on the cluster were both executed using four datasets and three kinds of viewpoints.

(2) To compare the pre-processing time and the calculation time of ArcGIS with those of the cluster, the respective time overheads in two environments described in experiment (1) were recorded.

(3) To verify the calculation accuracy of the Spark-based XDraw algorithm, the R3 algorithm implemented on the workstation was chosen as a reference. The viewshed analyst tool of ArcGIS on the workstation and the XDraw algorithm implemented on the cluster were both executed using four datasets and three kinds of viewpoints.

D. Performance Evaluation

The speedup ratio and accuracy ratio were used to measure the performance of the two proposed strategies. The speedup ratio is defined as the ratio between the computing time of the viewshed analysis executed not on the cluster to that of implemented on the cluster. Its equation is as follows:

$$S = \frac{T_{not\ on\ cluster}}{T_{on\ cluster}} \tag{7}$$

where S is the speedup ratio, $T_{not\ on\ cluster}$ is the execution time of the viewshed algorithm not on the cluster, and $Ton\ cluster$ is the computing time of the strategies being adopted on the cluster.

The accuracy ratio is defined as the ratio between the numbers of visible grid cells calculated by not using the R3 algorithm and those of using the R3 algorithm. Its equation is as follows:

$$AR = \frac{N_{not\ using\ R3}}{N_{using\ R3}} \quad (8)$$

where AR is the accuracy ratio, $N_{using\ R3}$ is the numbers of visible cells of the R3 algorithm, and $N_{not\ using\ R3}$ the numbers of visible cells calculated by other algorithms.

E. Results and Discussion

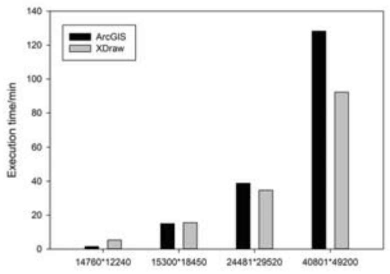

Fig. 8. Four datasets of different size on the pit viewpoint

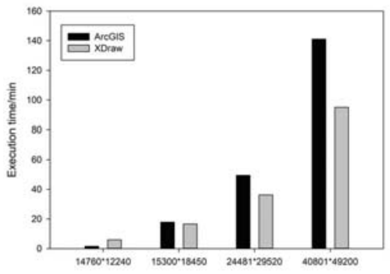

Fig. 9. Four datasets of different size on the peak viewpoint

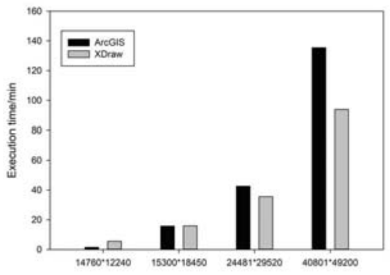

Fig. 10. Four datasets of different size on the flat viewpoint

1) Effectiveness of the strategy on parallel performance: The computing performance results shown in Fig.8-10 and TABLE V indicate that the computational efficiency of the Spark-based XDraw algorithm is improved significantly as the amount of data increases. Firstly, the use of the proposed tile-based storing strategy has a compounded influence on the acceleration of the XDraw algorithm, because the basic unit of storage and computation is a raster tile rather than a grid cell. This approach is more suitable for coarse-grained data access requirements and computing characteristics of the distributed framework Spark. Secondly, the influence of the amount of data on the performance of the algorithm cannot be ignored. A larger amount of input data corresponds to a higher potential speedup, because when the data size is relatively small, the time cost in job initialization for the distributed framework occupies a high percentage of the overall computing time. The initialization work includes allocating CPU, memory and other resources for each container before the task is started. Finally, the position of the viewpoint on the terrain directly affects the performance of the algorithm. In this experiment, when the viewpoint is a pit, the execution time of the XDraw algorithm is fewest under the conditions of three kinds of viewpoints. By contrast, when the viewpoint is a peak, the execution time of the algorithm is longer than that of the two other kinds. When the viewpoint is a pit, the visible areas are relatively small and the computational overhead is less. Instead, the visible areas are relatively large when the viewpoint is a peak, which increases the computational overhead as a result.

2) Time comparison of the pre-processing and calculation: The time spent on pre-processing cannot be ignored either in a single machine environment or under a Hadoop cluster. In this experiment, the main job of pre-processing in ArcGIS is to import raster data from Surfer grid file format to ESRI grid format. In contrast, the main job of the proposed storing strategy is to partition a raster file with Surfer grid file format into tile sets, and submit them to HDFS. The time-consuming results shown in TABLE VI show that with the increase of the amount of data, the pre-processing time of ArcGIS is nearing half of the calculation time in a single computer environment, and is much longer than that of the cluster. Instead, the time cost of tile partitioning is only a small portion of the computing time using the proposed strategy under Hadoop. Although the pre-processing time is not short, this process only needs to be done once. Once multiple input raster data layers have been pre-processed, they can be manipulated repeatedly with diverse terrain analysis algorithms, and the performance improvement is obvious.

3) Accuracy comparison of algorithms: In the experiment, the R3 algorithm was selected as a reference because of its superior accuracy. The differences of the three viewshed algorithms shown in TABLE VII. The accuracy of the Spark-based XDraw algorithm is higher than that of ArcGIS.

The accuracy ratio results listed in TABLE VIII show that overall computational accuracy of the Spark-based XDraw algorithm is relatively higher than that of ArcGIS for the 4 different data sets. Furthermore, the position of the viewpoint on the terrain has a direct influence on the accuracy of the calculation. When the viewpoint is a pit, the accuracy of ArcGIS is the lowest under the conditions of three kinds of

TABLE V
RESULTS OF THE COMPUTING TIME AND SPEEDUP RATIOS (TIME UNIT: MIN)

DataSet	Viewpoint	ArcGIS	XDraw	Speedup ratio
Grid1	pit	1.45	5.27	0.28
	peak	1.50	5.93	0.25
	flat	1.47	5.54	0.27
Grid2	pit	15.00	15.57	0.96
	peak	17.87	16.56	1.08
	flat	15.77	15.92	0.99
Grid3	pit	38.83	34.73	1.12
	peak	49.45	36.24	1.36
	flat	42.58	35.56	1.20
Grid4	pit	128.20	92.24	1.39
	peak	141.08	95.092	1.48
	flat	135.45	93.91	1.44

TABLE VI
TIME COMPARISON OF THE PRE-PROCESSING AND CALCULATION (TIME UNIT: MIN)

Dataset	Viewpoint	ArcGIS		XDraw	
		Pre-processing	Calculation	Pre-processing	Calculation
Grid1	pit	4.56	1.45	0.74	5.27
	peak	4.56	1.50	0.74	5.93
	flat	4.56	1.47	0.74	5.54
Grid2	pit	7.58	15.00	1.02	15.57
	peak	7.58	17.87	1.02	16.56
	flat	7.58	15.77	1.02	15.92
Grid3	pit	20.67	38.83	2.85	34.73
	peak	20.67	49.45	2.85	36.24
	flat	20.67	42.58	2.85	35.56
Grid4	pit	63.23	128.20	9.89	92.24
	peak	63.23	141.08	9.89	95.092
	flat	63.23	135.45	9.89	93.91

TABLE VII
THE NUMBERS OF VISIBLE GRID CELLS OF THREE ALGORITHMS

Dataset	Viewpoint	R3	ArcGIS	XDraw
Grid1	pit	4121792	3952801	4112734
	peak	19752064	19149280	19425786
	flat	6968482	6711822	6874146
Grid2	pit	6072126	5797605	6050279
	peak	29296093	28369874	28800712
	flat	10278537	9884614	10130983
Grid3	pit	13729891	13082838	13661857
	peak	67635096	65440991	66479243
	flat	23530907	22592941	23181167
Grid4	pit	33193997	31527392	33000215
	peak	167789113	162238136	164875294
	flat	57672268	55277329	56844918

viewpoints and the accuracy of the XDraw algorithm is the highest. The accuracy of the XDraw algorithm is 4% higher than that of ArcGIS on the whole. By contrast, when the viewpoint is a peak, the accuracy of the XDraw algorithm is the lowest under the three conditions, and the accuracy of ArcGIS is the highest. In this case, the accuracy of the XDraw algorithm is only 1.5% higher than that of ArcGIS. When the viewpoint is a pit, the invisible areas are relatively large and errors caused by the algorithm are reduced accordingly. Instead, the invisible areas are relatively small when the viewpoint is a peak, which leads to errors in the algorithm are increased as a result.

VI. CONCLUSION AND FUTURE WORK

Viewshed analysis is an important tool in the study of the digital terrain model. The XDraw algorithm can be achieved desirable computing performance for analyzing large terrain datasets by exploiting the distributed computing capability of Apache Spark. In this article, we presented both a tile-based storing strategy and an equal-volume computing s-trategy for parallel computation of the XDraw algorithm. The basic idea of the proposed tile-based storing strategy is to take a raster tile as the minimum process unit on the calculation phases to reduce the data transferring overhead of the distributed computation. Meanwhile, the equal-volume

TABLE VIII
ACCURACY COMPARISON OF THE ARCGIS AND THE XDRAW ALGORITHM

DataSet	Viewpoint	Accuracy ratio	
		ArcGIS	XDraw
Grid1	pit	95.90%	99.78%
	peak	96.95%	98.35%
	flat	96.32%	98.65%
Grid2	pit	95.48%	99.64%
	peak	96.84%	98.31%
	flat	96.17%	98.56%
Grid3	pit	95.29%	99.50%
	peak	96.76%	98.29%
	flat	96.01%	98.51%
Grid4	pit	94.98%	99.42%
	peak	96.69%	98.26%
	flat	95.85%	98.57%

computing strategy is benefit for achieving efficient load balancing to improve the parallel performance of the viewshed analysis. The experimental results showed that the Spark-based XDraw algorithm dramatically reduced the computing time and achieved satisfactory speedups, and it had higher computational efficiency and accuracy than the commercial GIS software.

Due to the lack of high-resolution elevation data, the elevation dataset of Australia with 30m spatial resolution was used to test the parallel performance of the Spark-based XDraw algorithm. In the future, we plan to collect additional high-resolution data and conduct assessment of the two strategies using real-world geospatial applications. Furthermore, the performance of the Spark-based XDraw algorithm needs to be improved. Through the experiment, we found that the overhead of the I/O operation of the viewshed analysis was much higher than that of the calculation. The input overhead was mainly derived from the frequent reading operations, and the data combination and write-back operations of the viewshed results contributed to most of the output overhead. In our future work, we will also investigate how to further optimize our strategies to address these problems. In addition, whether the proposed strategies have a suitable adaptability to other global operators of map algebra, such as surface or hydrologic analysis, would be verified through follow-up experiments.

ACKNOWLEDGMENT

This study was partially supported by the National Natural Science Foundation of China under grant No.41871304, and the Fundamental Research Funds for the Central Universities of China University of Geosciences (Wuhan) under grant No.CUG2018JM14.

REFERENCES

[1] Y. Zhao, A. Padmanabhan, S. Wang, A parallel computing approach to viewshed analysis of large terrain data using graphics processing units, International Journal of Geographical Information Science 27 (2) (2013) 363–384.

[2] H. M. Dodd, The validity of using a geographic information system's viewshed function as a predictor for the reception of line-of-sight radio waves, Ph.D. thesis, Virginia Tech (2001).

[3] S. V. Magalhaes, M. V. Andrade, W. R. Franklin, Multiple observer siting in huge terrains stored in external memory, International Journal of Computer Information Systems and Industrial Management (IJCISIM) 3 (2011) 143–149.

[4] W. R. Franklin, C. Ray, Higher isnt necessarily better: Visibility algorithms and experiments, in: Advances in GIS research: sixth international symposium on spatial data handling, Vol. 2, Taylor & Francis Edinburgh, 1994, pp. 751–770.

[5] J. Wang, G. J. Robinson, K. White, A fast solution to local viewshed computation using grid-based digital elevation models, Photogrammetric Engineering and Remote Sensing 62 (10) (1996) 1157–1164.

[6] Y. Zhi, L. Wu, Z. Sui, H. Cai, An improved algorithm for computing viewshed based on reference planes, in: 2011 19th International Conference on Geoinformatics, IEEE, 2011, pp. 1–5.

[7] N. Stojanović, D. Stojanović, Performance improvement of viewshed analysis using gpu, in: 2013 11th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services (TELSIKS), Vol. 2, IEEE, 2013, pp. 397–400.

[8] T. Axell, M. Fridén, Comparison between gpu and parallel cpu optimizations in viewshed analysis, Ph.D. thesis, Masters thesis in Computer Science: Algorithms, Languages and Logic (2015).

[9] L. Toma, Viewsheds on terrains in external memory, Sigspatial Special 4 (2) (2012) 13–17.

[10] A. Osterman, L. Benedičič, P. Ritoša, An io-efficient parallel implementation of an r2 viewshed algorithm for large terrain maps on a cuda gpu, International Journal of Geographical Information Science 28 (11) (2014) 2304–2327.

[11] M. V. Larsen, Viewshed algorithms for strategic positioning of vehicles, Master's thesis (2015).

[12] A. J. Cauchi-Saunders, I. J. Lewis, Gpu enabled xdraw viewshed analysis, Journal of Parallel and Distributed Computing 84 (2015) 87–93.

[13] X. Song, G. Tang, X. Liu, W. Dou, F. Li, Parallel viewshed analysis on a pc cluster system using triple-based irregular partition scheme, Earth Science Informatics 9 (4) (2016) 511–523.

[14] Z. Xu, Q. Yao, A novel algorithm for viewshed based on digital elevation model 2 (2009) 294–297.

[15] J. C. C. Bravo, T. Sarjakoski, J. Westerholm, Efficient implementation of a fast viewshed algorithm on simd architectures (2015) 199–202.

[16] K. Mills, G. Fox, R. Heimbach, Implementing an intervisibility analysis model on a parallel computing system, Computers and Geosciences 18 (8) (1992) 1047–1054.

[17] Y. Wang, S. Wang, Research and implementation on spatial data storage and operation based on hadoop platform, in: 2010 Second IITA International Conference on Geoscience and Remote Sensing, Vol. 2, IEEE, 2010, pp. 275–278.

[18] S. Ladra, J. R. Paramá, F. Silva-Coira, Scalable and queryable compressed storage structure for raster data, Information Systems 72 (2017) 179–204.

[19] D. Borthakur, The hadoop distributed file system: Architecture and design, Hadoop Project Website 11 (2007) 21.