Spark-Enabled XDraw Viewshed Analysis

Jianbo Zhang, Subin Zhao, and Zhuangzhuang Ye

Abstract—**Viewshed analysis is an indispensable part of digital terrain analysis and widely used in many application domains. High-resolution raster digital elevation model (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 visibility calculation can be accelerated using Apache Spark. In this article, we present a Spark-based parallel computing approach for the XDraw algorithm, which is composed of a tile-based raster data storing strategy, an equivolume computing strategy, and a stream-merging write-back strategy. The parallel implementation of the XDraw algorithm mainly consists of three parts: partitioning a raster DEM file into square tile sets and reorganizing these tile sets to prevent tile overlap across data divisions of Hadoop Distributed File System, subdividing the DEM into multiple equivolume data sectors according to the viewpoint position, and performing the XDraw algorithm on the corresponding tile sets of each sector independently and writing back the viewshed results efficiently. Experiments on real-world datasets show that the proposed computing approach can achieve higher speedup and efficiency for XDraw viewshed analysis as the raster DEM data volume is dramatically increased. The results also show that the approach has also satisfactory scalability as the number of data nodes in clusters is increased.**

Index Terms—**Parallel computing, Spark, viewshed analysis, XDraw.**

I. Introduction

IN RECENT years, Big Earth Data Analytics becomes necessary for applying the rapidly increasing amount of Big Earth Data [1], [2] to scientific research works and social lives. It can discover patterns, correlations, principles, knowledge, and other information for better responding to problems induced by global and regional changes [3]. Viewshed analysis is a terrain-based spatial modeling method, which can be performed on digital elevation models (DEMs) to determine areas visible from one or multiple specified observation viewpoints. It is a typical case of Big Earth Data Analytics when dealing with big terrain data. It has been widely used in scenic path planning [4], military monitoring [5], locations selection for telecommunication facilities [6], classification of cityscape areas [7], visibility of oceanic blue space [8], forest practices planning [9], and some other fields.

Manuscript received November 5, 2020; revised December 19, 2020; accepted January 7, 2021. Date of publication January 12, 2021; date of current version February 1, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 41871304. *(Corresponding author: Jianbo Zhang.)*

The authors are with the School of Geography Information Engineering, China University of Geosciences, Wuhan 430074, China (e-mail: zjb_tigers@126.com; lostshy0@163.com; 963372120@qq.com).

Digital Object Identifier 10.1109/JSTARS.2021.3051210

The volume of digital terrain data has reached to an unprecedented scale with the development of the data collection techniques (such as LiDAR). For example, the global SRTM1 data with a resolution of one arc-second in latitude and longitude have 10^{11} points [10]. However, mature commercial geographic information system (GIS) software (such as ArcGIS) cannot perform the viewshed analysis on this huge volume of data due to its compute-intensive execution. A computing approach to improve the computational efficiency of the viewshed algorithm is necessary. Parallel and high-performance computing (HPC) has persisted as a fundamental challenge for GIS [10]. It shows tremendous potential in advancing parallel spatial data processing [11]–[14].

The basic calculation procedure of the viewshed analysis is implemented by a line-of-sight (LoS) algorithm implemented on a raster DEM data [15], which calculates the visibility of a specific location through emanating a ray from the viewpoint to the target. It will cause a lot of repeated calculation if each ray from the target viewpoint to the grid cell of the DEM is calculated independently. The existing viewshed algorithms can be classified into two categories according to whether the intermediate calculation results are reused: nonreusable algorithm and reusable algorithm.

The nonreusable algorithm is relatively accurate but time consuming due to repeated calculations. The time complexity of this algorithm is $O(n^3)$, where n is the number of topographic points of the raster DEM. The R3 algorithm is a typical example of the nonreusable algorithm [16]. It calculates the visibility of each target cell by running a separate LOS emanating from the viewpoint to every target cell on the grid terrain [16]–[19] It is slow but excellent in accuracy. Although some improved algorithms for the R3 were proposed to improve the computational efficiency [20]–[22], there is still a problem for processing large amounts of data.

The reusable algorithm reuses intermediate calculation results to save the time cost of repeated computations but loss of accuracy. The R2 algorithm and the XDraw algorithm are both the reusable algorithms and run on $O(n^2)$ time. The R2 algorithm belongs to the reusable approximation point method, which reduces the complexity by calculating the intersection of the sight line and the DEM grid boundary by reusing the intermediate calculation results. It is faster but lower in accuracy [23] despite some improvements, such as the radarlike algorithm [24]. The XDraw algorithm belongs to the reusable point-by-point outward method [25]. Instead of performing the LoS calculation to any given point on the DEM and then progressing to its neighboring points, it emanates from the observer position on the DEM and gathers the LoS information, which can reduce the calculation time compared with the R3 algorithm and gain better calculation accuracy compared with the R2 algorithm. Meanwhile, the spatial independency of the XDraw algorithm makes it possible to optimize the algorithm to attain a high level of parallelization. Therefore, the XDraw algorithm was chosen in our implementation.

In the last decade, HPC technology has been used to accelerate these two category algorithms when massive terrain data are treated. Most research has focused on the design and parallel implementation of different viewshed analysis algorithms based on general-purpose computation on graphics processing units (GPGPUs) [26]. For the R3 algorithm, it can gain good performance improvement through a proper spatial domain decomposition strategy to achieve efficient I/O management [27], or a parallel implementation using GPU-based architecture [28]. For the R2 algorithm, some parallel implementations based on the CUDA programming framework were presented to obtain better computing efficiency [29], [30]. For the XDraw algorithm, many research efforts focused on reducing I/O costs on large datasets, including the scan-line filling method [31] and the minimum circumscribed rectangle of polygon method [32]. These efforts also include improving the parallel efficiency and the computational accuracy of the algorithm using GPUs [33] and optimizing the algorithm to improve the computational efficiency for massive DEM data [34].

In the same period, distributed computing frameworks, such as Apache Spark, have emerged. Compared with the expensive GPU clusters, it could organize and utilize currently idle computers to provide cheap and good quality computing service. Resilient distributed dataset (RDD) of Spark can emulate any distributed computing, and can give applications control of the most common bottleneck resources in clusters, such as storage I/O, which enable RDD to support effective emulation in mostly situations and to optimize efficiency issues caused by I/O problems [35]. This is the reason why Spark is rapidly becoming widely used as a fast and general-purpose cluster computing framework for large-scale data processing, such as power grid data [33] or spatial data [36]. Meanwhile, the cloud computing framework also provides a good solution for raster-based big data processing [37]–[39]. The research effort on evaluating the performance of the existing viewshed algorithms on cloud clusters using Spark has proved the possibility of distributed geospatial computing [40]. However, little work has been done to develop a Spark-based parallel computing approach for the XDraw algorithm to achieve high performance.

This article aims to present a parallel computing approach to implement the XDraw algorithm using Spark. 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 to prevent tile overlap across data divisions of Hadoop Distributed File System (HDFS).
2) Subdividing the DEM into multiple equivolume data sectors according to the given viewpoint position.
3) Retrieving the corresponding tile sets of each sector to perform the XDraw algorithm, merging and writing back the viewshed results.

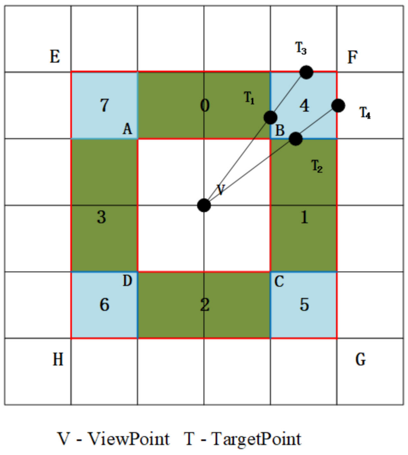

Fig. 1. Principle of the XDraw algorithm.

The major original contributions of this article include the following.

1) A tile-based data storing strategy is proposed to solve the problem of utilizing the HDFS to manage raster data.
2) An equivolume computing strategy for the distributed framework Spark is designed to achieve parallel acceleration for the XDraw algorithm.
3) A stream-merging write-back strategy is introduced to deal with persistent storage of massive viewshed results.

The remainder of this article is organized as follows. In Section II, the XDraw sequential algorithm for viewshed analysis is illustrated, and both the principle of the three proposed strategies and the implementation of the parallel algorithm are described in detail. In Section III, the computational experiments that are designed to evaluate the performance and the accuracy of the Spark-based approach are presented. In Section IV, the experimental results are presented and discussed. Conclusions and future work are presented in Section V.

II. Methods

A. *XDraw Sequential Algorithm*

The basic idea of the XDraw viewshed algorithm is to obtain the visibility of each grid cell through diffusing out from the observer to its neighboring points and reusing the LoS calculation results of them. It 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, as shown in Fig. 1. Given a viewpoint V in the raster DEM, the algorithm takes it as a center and diffuses outward in eight directions to form an initial ring. Then, gradually diffuses out and reuses the visibility of neighboring points to calculate the visibility of each point on the ring. This computational process continues until the boundary of the DEM is reached. The specific calculation process of the algorithm is described as follows.

First, the initial ring R_0 (rectangle ABCD) is obtained by diffusing the viewpoint V outward by a grid cell in eight directions (45° octants in each direction). Second, the initial ring

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: $Output\ the\ binary\ raster\ data$
```

R_0 (rectangle ABCD) is diffused outward to get a new ring R_1 in eight directions with each step of a grid cell. Then, the ring R_1 is divided into eight 45° octants to form rectangles 0–7. The visibility of points lying in the ring R_1 is calculated based on the visibility of points on the ring R_0 and the elevation differences between the points on the rectangle ABCD and the viewpoint V. The computing sequence of the rectangles is 0, 1, 2, 3, 4, 5, 6, and 7. By repeating the calculation steps above, the visibility of points lying in the R_{i+1} can be deduced from the visibility of points on the ring R_i and the elevation differences of the points on the ring R_i to the viewpoint V. The algorithm ends when the boundary of the DEM is reached. More details of the XDraw algorithm are described as follows.

In the aforementioned steps, steps 1 and 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. Step 17 outputs the binary visibility data, in which the cell value 1 represents visible, value -1 represents invisible, and value 0 represents an invalid value.

By analyzing the implementation of XDraw sequential algorithm, we can find out that: first, the XDraw sequential algorithm could be accelerated by means of the power of parallel computing. The eight subrectangles formed in each iterative 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 when executing. Second, the equiangular data partitioning strategy adopted by the original XDraw algorithm, which could cause load imbalance during the distributed computing process. The location of the viewpoint may cause huge differences of the

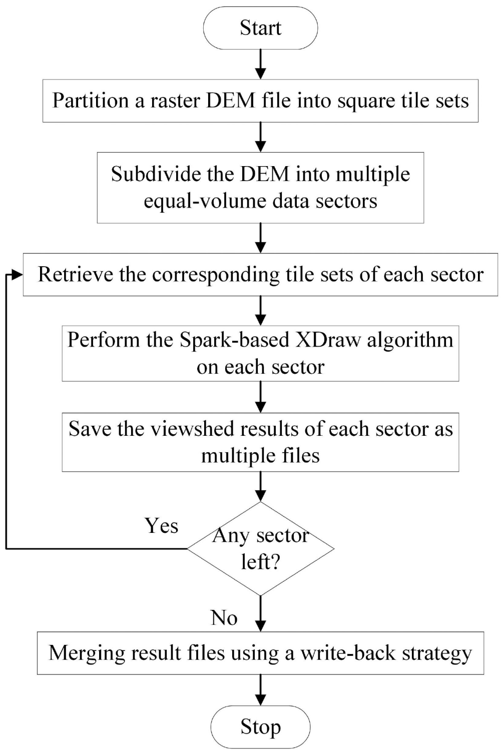
Start
Partition a raster DEM file into square tile sets
Subdivide the DEM into multiple equal-volume data sectors
Retrieve the corresponding tile sets of each sector
Perform the Spark-based XDraw algorithm on each sector
Save the viewshed results of each sector as multiple files
Any sector left?
Yes
No
Merging result files using a write-back strategy
Stop

Fig. 2. Flowchart of the XDraw algorithm on Spark.

data volume in the eight subrectangles. For instance, when the viewpoint position is not in the central area of the DEM (such as on the boundary), the data amount of each subrectangle varies greatly, which could lead to a distinctly different amount of data allocated to each computing node in the cluster environment. The resulting load imbalance of the distributed computing will reduce the efficiency of parallel computing. In order to solve this problem, this article proposed an equivolume computing strategy to improve the parallel computing performance of the Spark-based XDraw algorithm effectively.

B. Principle of Spark-Based Computing Approach

The distributed XDraw algorithm combines a tile-based raster data storing strategy, an equivolume computing strategy, and a stream-merging write-back strategy on Spark for a raster DEM data. First, the DEM data are partitioned into square tile sets before submitted to HDFS. Second, the DEM data are logically subdivided into multiple equivolume sectors according to the position of the viewpoint. Then, the Spark-based XDraw algorithm is performed on each sector and their viewshed results are saved as multiple separate files. Finally, these files are merged into an entire file using a write-back strategy. The flowchart of the proposed implementation of the XDraw algorithm on Spark is illustrated in Fig. 2.

1) Tile-Based Raster Data Storing Strategy: Hadoop is an open-source software that could utilize currently idle computers and storage resource to provide economic and high-quality computing service. HDFS is suitable for applications with high-throughput access requirements for large datasets. The reason

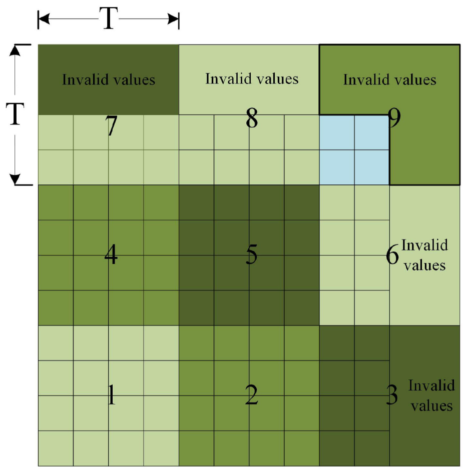

Fig. 3. Tile decomposition strategy.

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 [41], 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 [42]. The size of a tile is typically smaller than that of a data block. Consequently, multiple raster tiles can be reorganized 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. A HDFS file is divided into 64-MB partitions. This division method can make a computation requested by an application be executed near the data it operates on as far as possible. It helps minimize network congestion and increase overall system throughput [43]. 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, which could bring out additional communication overhead. 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 proposed [44]. 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 following two steps.

1) Partition the raster data into a sequence of tile sets with equal rows and columns, including auxiliary metadata information.
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.

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 in Fig. 3. The raster

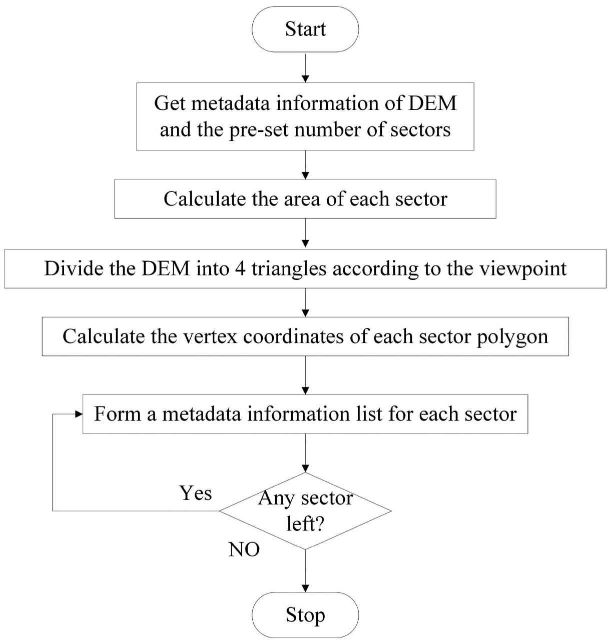
Start
Get metadata information of DEM and the pre-set number of sectors
Calculate the area of each sector
Divide the DEM into 4 triangles according to the viewpoint
Calculate the vertex coordinates of each sector polygon
Form a metadata information list for each sector
Yes
Any sector left?
NO
Stop

Fig. 4. Flowchart of the equal-volume computing strategy.

data are divided into multiple square tiles with a specified tile size T

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

where Row and Col denote the row and column numbers of the raster data, respectively. 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 $-99\,999$) to ensure all tiles have the same size. The tile size should be determined dynamically according to the volume of raster data. In most commercial GIS software, the byte length of a raster cell is often multiples or divisors of 4 B. For instance, in ArcGIS, a pixel consists of 4-B floating point or integer-type data. Therefore, 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 thus avoided.

2) Equal-Volume Computing Strategy: 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 equivolume computing strategy was proposed to deal with this problem. The main idea is to adopt an equivolume condition as a criterion for logical partitioning of the DEM.

A raster DEM 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 equivolume sectors according to the position of the viewpoint and the preset number of sectors. The crucial step of the equivolume 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 equivolume computing strategy is illustrated in Fig. 4.

TABLE I
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	$X = 0$	$Y = Row - 1 - ((N_i * S - S_{VAB} - S_{VBC} - S_{VCD}) * 2/X)$

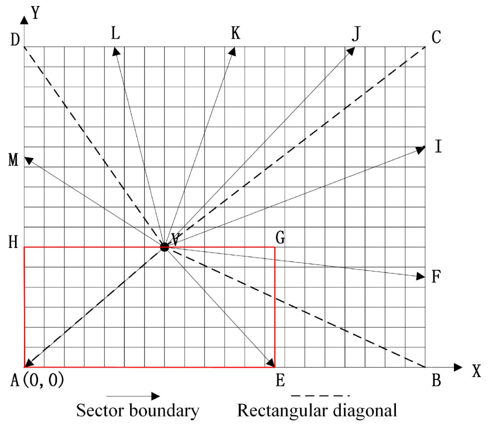

Fig. 5. Equal-volume partitioning strategy.

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 preset number of sectors

$$S = (\text{Row} * \text{Col})/N \tag{2}$$

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

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

3) Calculate the vertex coordinates of each sector polygon. The calculation starts with line VA and is carried out anticlockwise. 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 four basic triangles (triangles VAB, VBC, VCD, and VDA composed of dotted lines shown 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.

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

4) Generate a metadata information list for each sector. The 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 MBR AEGH of the sector VAE is obtained by comparing the minimum and maximum values of its three vertex coordinates. By repeating the steps above, a metadata information list for each sector is formed in terms of the following format described in Table II.

3) Stream-Merging Write-Back Strategy: In addition to data organization and computation, the persistent storage of the viewshed results is also an important issue. After each sector has performed the Spark-based XDraw algorithm, its viewshed results will be transformed from key-value RDDs to byte array to save in a separate file, and then these files are merged into a whole file on HDFS. This work needs to be done by launching a Spark executor. An executor is responsible for carrying out the tasks making up the Spark application and returning the results to a Spark driver. In order to implement the conversion and merging of viewshed results, each sector must allocate a chunk of memory of the same size as the product of the rows and columns of the raster DEM. When the memory space size required by all sectors exceeds that of the current executor, the executor fails to deal with massive output viewshed results. Therefore, a proper write-back strategy on Spark should be concerned.

A stream-merging write-back strategy was proposed to address this problem. Its main idea is to first allocate a memory space with equal size to that of the viewshed results of each sector, and then calculate the global position of each valid grid cell relative to the raster DEM and finally write the viewshed results of each sector into a complete result file on HDFS. There is a crucial step that needs to be considered in the implementation of the write-back strategy. The number of a grid cell in a sector is a local value calculated in terms of the vertex coordinates of each sector polygon. It should be converted into a global offset according to metadata information of the sector before its viewshed results are written to the final result file. The flowchart of the proposed write-back strategy is illustrated in Fig. 6.

The implementation of this strategy includes the following four steps.

1) Read the columns of tiles in the DEM and the preset number of sectors from DEM metadata information.
2) Obtain the MBR from the metadata information of each sector.
3) Read grid cells in the sector in turn, and calculate the global offset for each valid cell in the final result file based on the

TABLE II
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 sector's MBR	8
RectangleEnd	Point(X,Y)	The upper right corner of the sector's MBR	8

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

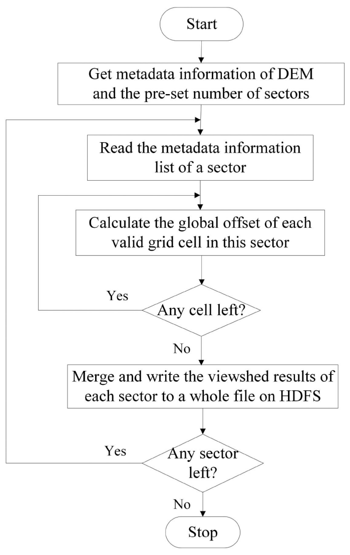
Start
Get metadata information of DEM and the pre-set number of sectors
Read the metadata information list of a sector
Calculate the global offset of each valid grid cell in this sector
Yes
Any cell left?
No
Merge and write the viewshed results of each sector to a whole file on HDFS
Yes
Any sector left?
No
Stop

Fig. 6. Flowchart of stream-merging write-back strategy.

following formula:

$$\begin{aligned}\text{index} = {} & [\text{RectangleStart}.Y \\ & + i/(\text{RectangleEnd}.X - \text{RectangleStart}.X)] \\ & \times \text{TileColNum} + [\text{RectangleStart}.X \\ & + i\%(\text{RectangleEnd}.X - \text{RectangleStart}.X)]\end{aligned} \tag{3}$$

where i denotes the number of a grid cell in a sector, TileColNum denotes columns of tiles illustrated in Table III, and RectangleStart and RectangleEnd denote the left lower and upper right corner of the sector's MBR illustrated in Table II, respectively.

4) Merge and write the viewshed results each sector into a complete result file on HDFS, and loop till all sectors have been processed.

C. Algorithm Implementation

1) Partition Algorithm for Raster Data 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 using a predefined tile size (TileSize = 256)

$$\text{TileRowNum} = \begin{cases}\text{RowNum/TileSize} + 1 \\ \quad \text{if (RowNummodTileSize} \neq 0) \\ \text{RowNum/TileSize} \\ \quad \text{if (RowNummodTileSize} = 0)\end{cases} \tag{4}$$

$$\text{TileColNum} = \begin{cases}\text{ColNum/TileSize} + 1 \\ \quad \text{if (ColNummodTileSize} \neq 0) \\ \text{ColNum/TileSize} \\ \quad \text{if (ColNummodTileSize} = 0)\end{cases}. \tag{5}$$

Then, generate a metadata file according to this information to provide sufficient information for subsequent implementations. 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.

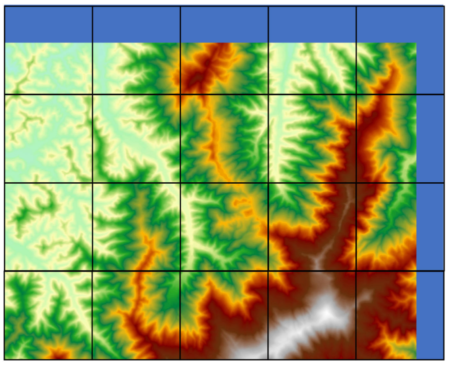

Fig. 7. Partitioning method for georaster.

1) Create a tile dataset file, and partition the raster file in terms of the method shown in Fig. 7.
2) Calculate the rows and columns of invalid values added

$$\text{AddRowNum} = \begin{cases} \text{TileRowNum} * \text{TileSize} - \text{RowNum} \\ \quad \text{if } (\text{RowNum mod TileSize} \neq 0) \\ 0 \\ \quad \text{if } (\text{RowNum mod TileSize} = 0) \end{cases} \tag{6}$$

$$\text{AddColNum} = \begin{cases} \text{TileColNum} * \text{TileSize} - \text{ColNum} \\ \quad \text{if } (\text{ColNum mod TileSize} \neq 0) \\ 0 \\ \quad \text{if } (\text{ColNum mod TileSize} = 0) \end{cases} \tag{7}$$

3) For each tile, perform the following steps.
   1) 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 by the number of AddColNum to form a whole tile.
   2) 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 by the number of AddRowNum to form a whole tile.
   3) 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, whereas the tile dataset file is distributed throughout all data nodes.

2) Equivolume 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 equivolume sectors using the proposed strategy described in the part 2 of Section II-B. More details of the algorithm are described in Algorithm 2.

In the aforementioned steps, steps 1–3 carry out the equivolume division on the DEM to obtain multiple sectors and their metadata information list. In Steps 4–17, a series of RDD operations are performed on each sector to get its RDD⟨index, value⟩ 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 (tile RDD) in step 8 represents the calculation procedure of the global index of each grid cell.

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 preset
    number of sectors
2:  get sectors using the equal-volume computing strategy
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:   $rectangle\ RDD\langle index, value\rangle \leftarrow$
      $union(real\ RDD)$
11:   for each $cell\langle index, value\rangle \in rectangle\ RDD$ 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:   $sector\ RDD\langle index, value\rangle \leftarrow$
      $rectangleRDD.filter(index > 0)$
17: end for
18: $Output\ the\ sector\ RDD\langle index, value\rangle$
```

.

3) XDraw Algorithm Based on Spark: Each sector performs the Spark-based XDraw algorithm. The viewshed results of all sectors are merged using the proposed write-back strategy to generate the visibility raster data saved in HDFS. More details of the implementation of the Spark-based XDraw algorithm are described in Algorithm 3.

In the aforementioned steps, steps 1–23 carry out the XDraw viewshed algorithm on the RDD of each sector, and save the sector's viewshed results as a separate file on HDFS. Steps 24–33 merge and write back the viewshed results of all sectors into a whole file on HDFS. Step 34 outputs the visibility rater data.

III. Experiments and Results

A. Datasets

The performance and accuracy of the proposed computing approach in this article were evaluated using the DEM of Australia.[1] To determine the impact of varying the data volume on the parallel performance, four elevation datasets of different size were used to test the effectiveness of the proposed approach via data resampling. Meanwhile, to determine the variation

[1][Online]. Available: https://data.gov.au/data/dataset/da926e47-1cd8-4dc9-b859-cbc18c29d858

Algorithm 3: Spark-Based XDraw Algorithm.

```
Input: sector $RDD$, the viewpoint $V$
Output: the visibility raster $B$
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:   write the sectors viewshed results to a file
23: end for
24: get metadata information of DEM and the pre-set number of sectors
25: for each $F_i \in sector\ result\ files(F)$ do
26:   read $F_i$ and get the metadata information list of a sector
27:   for each $cell \in sector\ cells(C)$ do
28:     if $the\ cell\ is\ visible$ then
29:       calculate the global offset of the cell
30:     end if
31:   end for
32:   Merge and write the sectors viewshed results to the final file
33: end for
34: $Output\ the\ visibility\ raster\ data$
```

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

The elevation map of Australia is shown in Fig. 8.

B. Hardware Environment

A computer cluster with two name nodes and eight data nodes was used as the hardware platform. These nodes are linked by 100-Mb/s Fast Ethernet. There are 57 cores in the compute

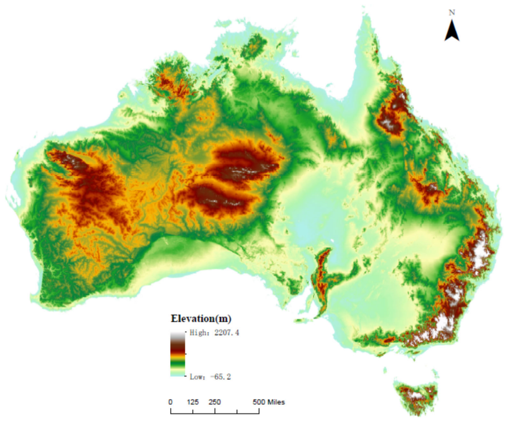

Fig. 8. Elevation map of Australia.

nodes of this cluster in all. The CentOS 6.5 operating system, Hadoop 2.6.0, and JDK 1.7 are used for each computer node. Meanwhile, a workstation with ArcGIS 10.1 installed was used to execute the experimental algorithm to compare preprocessing and calculation time with those of the cluster. It has 1 Intel(R) Xeon(TM) E3-1225 CPU, 12 GB of memory, and 1 TB of storage.

C. Experimental Designs

Five experiments were conducted to test the parallel performance of the three proposed strategies, which are as follows. 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 preprocessing time was also recorded to be compared with it.

1) To investigate the effectiveness of the proposed computing approach on parallel performance, both the viewshed analyst tool of ArcGIS on the workstation and the XDraw algorithm implemented on the cluster were executed using four datasets and three kinds of viewpoints.
2) To compare the preprocessing 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.
4) To test the scale-up performance of the proposed approach, the Spark-based XDraw algorithm was executed using four datasets and three kinds of viewpoints on two clusters with four and eight computational nodes, respectively.
5) To examine the effectiveness of the stream-merging write-back strategy, the Spark-based XDraw algorithm was carried out on four datasets and three kinds of viewpoints by adopting the strategy or not.

TABLE IV
AUSTRALIA ELEVATION DATASETS

Dataset	Size(GB)	Grid-cell size	Columns	Rows	Grid cells
Grid1	1.05	8”×8”	18450	15300	282285000
Grid2	2.69	5”×5”	29520	24481	722679120
Grid3	7.47	3”×3”	49200	40801	2007409200
Grid4	16.8	2”×2”	73800	61201	4516633800

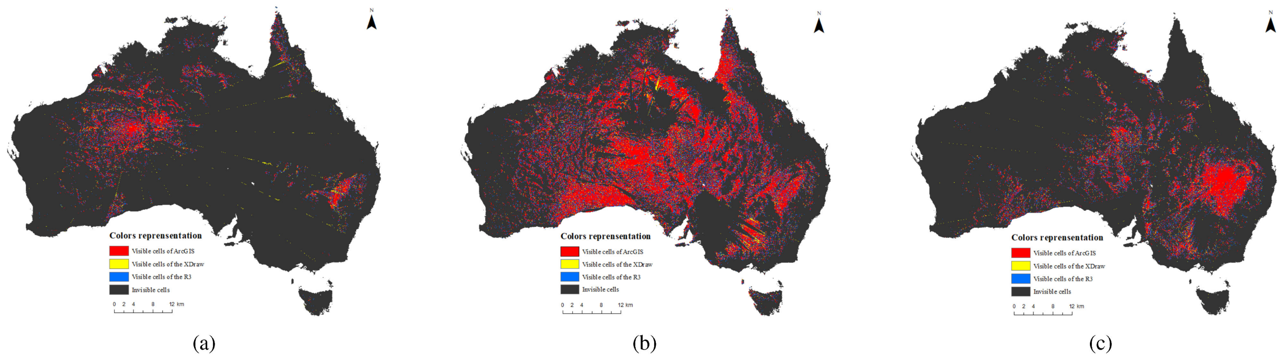
Colors representation
🟥 Visible cells of ArcGIS
🟨 Visible cells of the XDraw
🟦 Visible cells of the R3
⬛ Invisible cells
0 2 4 8 12 km
N
(a)
(b)
(c)

Fig. 9. Viewshed overlapping maps generated by three algorithms. (a) Pit viewpoint. (b) Peak viewpoint. (c) Flat viewpoint.

D. *Performance Evaluation*

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

$$S = \frac{T_{\text{not on cluster}}}{T_{\text{on cluster}}} \tag{8}$$

where S is the speedup ratio, $T_{\text{not on cluster}}$ is the computing time of the viewshed algorithm not on the cluster, and $T_{\text{on cluster}}$ is the computing time of the viewshed algorithm adopting the proposed strategies 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:

$$\text{AR} = \frac{N_{\text{not using R3}}}{N_{\text{using R3}}} \tag{9}$$

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

E. *Results*

The viewshed overlapping maps are shown in Fig. 9, which employs the R3, and the Spark-based XDraw and the ArcGIS algorithm implemented on the same dataset Grid3 and three different viewpoints. When computing such a large dataset as Grid4, the viewshed analyst tool of ArcGIS fails to obtain the viewshed results. Detailed experimental results and discussions are shown in Section IV.

IV. DISCUSSIONS

For all five experiments in this section, 256 × 256 is selected as the tile size. Except for the experiment shown in scale-up performance of the approach, which uses 4 and 8 computational nodes for comparative experiments, the other four experiments use eight computational nodes.

A. *Effectiveness of the Approach on Parallel Performance*

The computing performance results shown in Fig. 10 and Table V indicate that the computational efficiency of the Spark-based XDraw algorithm is improved significantly as the amount of data increases. First, 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. Second, 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. This is due to the time cost in job initialization for the distributed framework occupies a high percentage of the overall computing time when the data size is relatively small. 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. This is because the XDraw algorithm is based on the LoS algorithm and diffuses outward based on the visibility of adjacent points around the viewport. When the viewpoint is a pit, the visible areas are relatively small and the

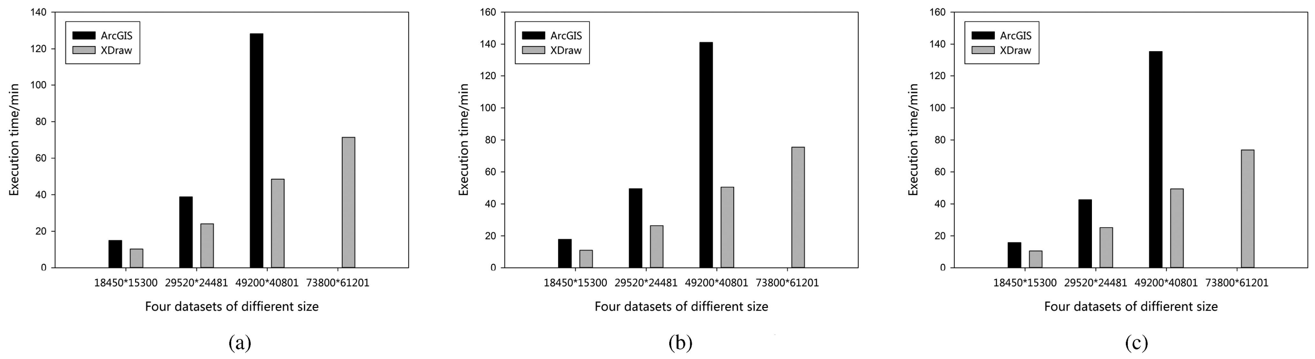

Fig. 10. Comparisons of the execution time of ArcGIS and the XDraw algorithm. (a) Pit viewpoint. (b) Peak viewpoint. (c) Flat viewpoint.

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

DataSet	Viewpoint	ArcGIS	XDraw	Speedup ratio
Grid1	pit	15.00	10.13	1.48
	peak	17.87	10.80	1.65
	flat	15.77	10.41	1.51
Grid2	pit	38.83	24.07	1.61
	peak	49.45	26.44	1.87
	flat	42.58	25.16	1.69
Grid3	pit	128.20	48.51	2.64
	peak	141.08	50.51	2.79
	flat	135.45	49.38	2.74
Grid4	pit	-	71.41	-
	peak	-	75.50	-
	flat	-	73.71	-

TABLE VI
TIME COMPARISON OF THE PREPROCESSING AND CALCULATION (TIME UNIT: MIN)

Dataset	Viewpoint	ArcGIS			XDraw		
		Pre-processing	Calculation	Proportion of pre-processing	Pre-processing	Calculation	Proportion of pre-processing
Grid1	pit	7.58	15.00	33.57%	0.74	10.13	6.81%
	peak	7.58	17.87	29.78%	0.74	10.80	6.41%
	flat	7.58	15.77	32.46%	0.74	10.41	6.64%
Grid2	pit	20.67	38.83	34.74%	1.02	24.07	4.07%
	peak	20.67	49.45	29.48%	1.02	26.44	3.71%
	flat	20.67	42.58	32.68%	1.02	25.16	3.90%
Grid3	pit	63.23	128.20	33.03%	2.85	48.51	5.55%
	peak	63.23	141.08	30.95%	2.85	50.51	5.34%
	flat	63.23	135.45	31.83%	2.85	49.38	5.46%
Grid4	pit	140.00	-	-	9.89	71.41	12.16%
	peak	140.00	-	-	9.89	75.50	11.58%
	flat	140.00	-	-	9.89	73.71	11.83%

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.

B. Time Comparison of the Preprocessing and Calculation

The time spent on preprocessing cannot be ignored either in a single machine environment or under a Hadoop cluster. In this experiment, the main jobs of preprocessing in ArcGIS include reading from HDD, partitioning data into blocks, performing pixel statistics, and compressing data. 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 indicate that with the increase of the amount of data, the preprocessing time of ArcGIS is nearing one-third of the entire 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 when using the proposed strategy under Hadoop. Although the preprocessing time is not short, this process only needs to be done once. Once multiple input raster data layers have been preprocessed, they can be manipulated repeatedly with diverse terrain analysis algorithms, and the performance improvement is obvious.

C. Accuracy Comparison of Algorithms

In the experiment, the R3 algorithm was selected as a reference because of its superior accuracy. The R3 algorithm calculates the visibility of each grid cell by running a separate

TABLE VII
NUMBERS OF VISIBLE GRID CELLS OF THREE ALGORITHMS

Dataset	Viewpoint	Number of visible grid cells			Accuracy ratio	
		R3	ArcGIS	XDraw	ArcGIS	XDraw
Grid1	pit	6072126	5797605	6051643	95.48%	99.66%
	peak	29296093	28369874	28920244	96.84%	98.72%
	flat	10278537	9884614	10153648	96.17%	98.78%
Grid2	pit	13729891	13082838	13674410	95.29%	99.60%
	peak	67635096	65440991	66652638	96.76%	98.55%
	flat	23530907	22592941	23206318	96.01%	98.62%
Grid3	pit	33193997	31527392	33018135	94.98%	99.47%
	peak	167789113	162238136	165130689	96.69%	98.42%
	flat	57672268	55277329	56876850	95.85%	98.62%
Grid4	pit	-	-	63077028	-	-
	peak	-	-	326552906	-	-
	flat	-	-	111419421	-	-

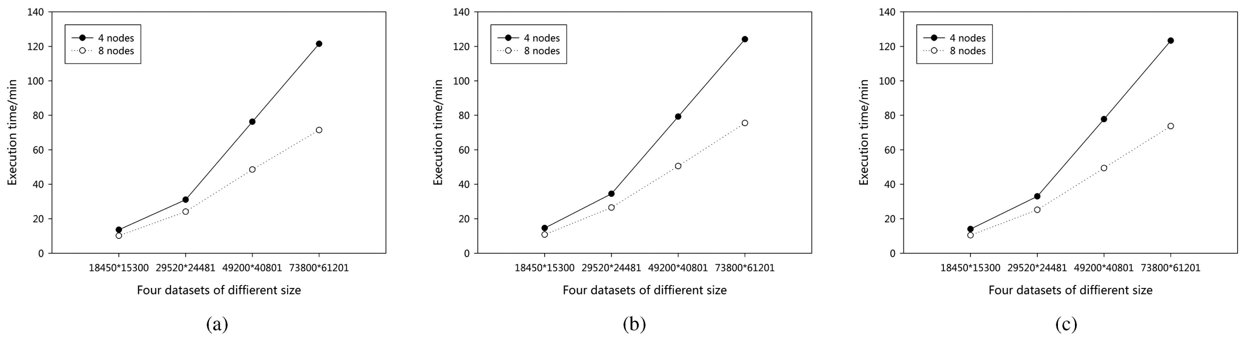

Fig. 11. Comparisons of the execution time of different number of computational nodes. (a) Pit viewpoint. (b) Peak viewpoint. (c) Flat viewpoint.

LoS emanating from the viewport to every grid cell on the raster DEM. Similarly, the R3 algorithm fails to work when dealing with dataset Grid4. In Table VII, the accuracy ratios of ArcGIS and the XDraw algorithm are calculated by using the numbers of visible grid cells of the R3 algorithm as a benchmark.

The accuracy ratio results listed in Table VII show that overall computational accuracy of the Spark-based XDraw algorithm is relatively higher than that of ArcGIS implemented on the first three datasets. 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 viewpoints, and the accuracy of the XDraw algorithm is the highest. The accuracy of the XDraw algorithm is 4.3% 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 among the three conditions, and that of ArcGIS is the highest. In this case, the accuracy of the XDraw algorithm is only 1.8% higher than that of ArcGIS. This is due to the principle and implementation of the XDraw algorithm. 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 increasing as a result.

D. *Scale-Up Performance of the Approach*

In this experiment, the Spark-based XDraw algorithm was carried out under the condition that the number of computational nodes is four and eight, respectively, due to the constraints of the cluster hardware environment, also using four datasets and three kinds of viewpoints. In Fig. 11, the horizontal axis represents four datasets of different size, and the two broken lines represent the number of computational nodes. The execution time results in Fig. 11 and Table VIII indicate that the proposed computing approach exhibits satisfactory scalability on Hadoop clusters. The computing time decreases stably as the number of computational nodes increases. It is because additional nodes improve task parallelism and distribute the raster tile sets more evenly on these nodes, which contribute to computation near storage under Spark.

E. *Effectiveness of the Stream-Merging Write-Back Strategy*

The computing performance results shown in Table IX indicate that the acceleration of XDraw algorithm under Spark also derives a lot of benefit from the use of stream-merging write-back strategy. First, the implementation based on the proposed strategy writes the viewshed results of each sector to multiple small files and then merge them into a large file. Compared with the conventional method that needs allocating a large memory space according to the rows and columns of the raster DEM, the write-back strategy has obvious advantages in processing big data. Second, a larger amount of output data corresponds to a higher potential speedup. It is because the time cost of job initialization and information exchange occupies a high percentage of the overall write-back time when the data size is relatively small. As the amount of data increases, the parallel efficiency employing the write-back method is improved significantly.

TABLE VIII
TIME COMPARISON OF DIFFERENT NUMBER OF COMPUTATIONAL NODES (TIME UNIT: MIN)

DataSet	Viewpoint	Number of computational nodes	
		4	8
Grid1	pit	13.63	10.13
	peak	14.61	10.8
	flat	14.01	10.41
Grid2	pit	31.04	24.07
	peak	34.48	26.44
	flat	32.98	25.16
Grid3	pit	76.24	48.51
	peak	79.22	50.51
	flat	77.76	49.38
Grid4	pit	121.5	71.41
	peak	124.13	75.5
	flat	123.31	73.71

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

Dataset	Viewpoint	Not adopted	Adopted	Speedup ratio
Grid1	pit	15.57	10.13	1.54
	peak	16.56	10.80	1.53
	flat	15.92	10.41	1.53
Grid2	pit	34.73	24.07	1.42
	peak	36.24	26.44	1.37
	flat	35.56	25.16	1.41
Grid3	pit	92.24	48.51	1.90
	peak	95.09	50.51	1.88
	flat	93.91	49.38	1.90
Grid4	pit	-	71.41	-
	peak	-	75.50	-
	flat	-	73.71	-

V. CONCLUSION

Viewshed analysis plays an important role in the study of the digital terrain model. Its execution process can become computationally intensive as data size increases. The XDraw algorithm can achieve desirable computing performance for analyzing large terrain datasets by exploiting the distributed computing capability of Apache Spark. In this article, we presented a Spark-based parallel computing approach for the XDraw algorithm, which integrated a tile-based raster data storing strategy, an equivolume computing strategy, and a stream-merging write-back strategy. First, the basic idea of the proposed tile-based storing strategy is to take a raster tile as the minimum process unit during the calculation phases to reduce the data transferring overhead of the distributed computation. Second, the equivolume computing strategy is beneficial for achieving efficient load balancing to improve the parallel performance of the viewshed analysis. Finally, the proposed write-back strategy contributes to deal with persistent storage of massive viewshed results. 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.

In future work, we will plan to collect additional high-resolution data and conduct assessment of the proposed computing approach using real-world geospatial applications. Then, we will conduct the experiment in a cluster, which is composed of better configured computational nodes. Furthermore, 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. We will also investigate how to further optimize our strategies to gain higher efficiency and accuracy. In addition, whether the proposed computing approach has a suitable adaptability to other global operators of map algebra, such as surface or hydrologic analysis, would be verified through follow-up experiments. The last but not the least, the experimental comparison with the GPU-based implementation was not presented in this article because of the huge differences between the two computational frameworks (Spark and GPGPU) in the principle and implementation methods. On the other hand, existing geodistributed computing frameworks presented in the literature (such as Hadoop-GIS, Spatial Hadoop, Spatial Spark, GeoSpark, and STARK) rarely involve the georaster analysis. We will continue to search for some suitable georaster computational frameworks and conduct quantitative comparisons with them.

V. ACKNOWLEDGMENT

The authors would like to thank D. Xia and Y. Li from the China University of Geosciences for their earlier research work.

REFERENCES

[1] H. Guo, Z. Liu, H. Jiang, C. Wang, J. Liu, and D. Liang, “Big Earth Data: A new challenge and opportunity for digital earth’s development,” *Int. J. Digit. Earth*, vol. 10, no. 1, pp. 1–12, 2017.

[2] M. Sudmanns *et al.*, “Big Earth Data: Disruptive changes in earth observation data management and analysis?,” *Int. J. Digit. Earth*, vol. 13, no. 7, pp. 832–850, 2020.

[3] S. Bhattacharyya and D. Ivanova, “Scientific computing and big data analytics: Application in climate science,” in *Distributed Computing in Big Data Analytics*. Berlin, Germany: Springer, 2017, pp. 95–106.

[4] J. L. D. Stucky, “On applying viewshed analysis for determining least-cost paths on digital elevation models,” *Int. J. Geograph. Inf. Sci.*, vol. 12, no. 8, pp. 891–905, 1998.

[5] M. V. Andrade, S. V. Magalhaes, M. A. Magalhaes, W. R. Franklin, and B. M. Cutler, “Efficient viewshed computation on terrain in external memory,” *Geoinformatica*, vol. 15, no. 2, pp. 381–397, 2011.

[6] J. Łubczonek, W. Kazimierski, and M. Pałczyński, "Planning of combined system of radars and CCTV cameras for inland waterways surveillance by using various methods of visibility analyses," in *Proc. 12th Int. Radar Symp.*, 2011, pp. 731–736.

[7] K. Czyńska and P. Rubinowicz, "Classification of cityscape areas according to landmarks visibility analysis," *Environ. Impact Assessment Rev.*, vol. 76, pp. 47–60, 2019.

[8] Y. Qiang, S. Shen, and Q. Chen, "Visibility analysis of oceanic blue space using digital elevation models," *Landscape Urban Planning*, vol. 181, pp. 92–102, 2019.

[9] K. Y. Lee, J. I. Seo, K.-N. Kim, Y. Lee, H. Kweon, and J. Kim, "Application of viewshed and spatial aesthetic analyses to forest practices for mountain scenery improvement in the Republic of Korea,*" Sustainability*, vol. 11, no. 9, 2019, Art. no. 2687.

[10] "The shuttle radar topography mission (SRTM) collection user guide," 2015. Accessed: Mar. 2020. [Online]. Available: https://lpdaac.usgs.gov/documents/179/SRTM_User_Guide_V3.pdf

[11] E. Shook *et al.*, "Parallel cartographic modeling: A methodology for parallelizing spatial data processing," *Int. J. Geograph. Inf. Sci.*, vol. 30, no. 12, pp. 2355–2376, 2016.

[12] Q. Guan and K. C. Clarke, "A general-purpose parallel raster processing programming library test application using a geographic cellular automata model," *Int. J. Geograph. Inf. Sci.*, vol. 24, no. 5, pp. 695–722, 2010.

[13] C. Yang, R. Raskin, M. Goodchild, and M. Gahegan, "Geospatial cyberinfrastructure: Past, present and future," *Comput., Environ. Urban Syst.*, vol. 34, no. 4, pp. 264–277, 2010.

[14] T. Cheng, J. Haworth, and E. Manley, "Advances in geocomputation (1996–2011)," *Comput., Environ., Urban Syst.*, vol. 36, no. 6, pp. 481–487, 2012.

[15] L. De Floriani, P. Marzano, and E. Puppo, "Line-of-sight communication on terrain models," *Int. J. Geograph. Inf. Syst.*, vol. 8, no. 4, pp. 329–342, 1994.

[16] W. R. Franklin and C. Ray, "Higher isn't necessarily better: Visibility algorithms and experiments," in *Proc. Adv. GIS Res., 6th Int. Symp. Spatial Data Handling*, 1994, vol. 2, pp. 751–770.

[17] M. Van Kreveld, "Variations on sweep algorithms: Efficient computation of extended viewsheds and class intervals," in *Proc. 7th Int. Symp. Spatial Data Handling*, 1996, pp. 13–15.

[18] A. Shapira, "Visibility and terrain labeling," Ph.D. dissertation, Dept. Elect., Comput., Syst. Eng., Rensselaer Polytech. Inst., Troy, NY, USA, May. 1990.

[19] D. Izraelevitz, "A fast algorithm for approximate viewshed computation," *Photogrammetric Eng. Remote Sens.*, vol. 69, no. 7, pp. 767–774, 2003.

[20] J. Wang, G. J. Robinson, and K. White, "A fast solution to local viewshed computation using grid-based digital elevation models," *Photogrammetric Eng. Remote Sens.*, vol. 62, no. 10, pp. 1157–1164, 1996.

[21] H. Wu, M. Pan, L. Yao, and B. Luo, "A partition-based serial algorithm for generating viewshed on massive DEMs," *Int. J. Geograph. Inf. Sci.*, vol. 21, no. 9, pp. 955–964, 2007.

[22] Y. Zhi, L. Wu, Z. Sui, and H. Cai, "An improved algorithm for computing viewshed based on reference planes," in *Proc. 19th Int. Conf. Geoinformat.*, 2011, pp. 1–5.

[23] J. Yu, J. Wu, and M. Sarwat, "GeoSpark: A cluster computing framework for processing large-scale spatial data," in *Proc. 23rd SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, 2015, pp. 1–4.

[24] M. V. Larsen, "Viewshed algorithms for strategic positioning of vehicles," Master's thesis, Norwegian Defence Res. Establishment (FFI). Univ. Oslo, Oslo, Norway, 2015.

[25] Z. Xu and Q. Yao, "A novel algorithm for viewshed based on digital elevation model," in *Proc. Asia-Pacific Conf. Inf. Process.*, 2009, vol. 2, pp. 294–297.

[26] T. Axell and M. Fridén, "Comparison between GPU and parallel CPU optimizations in viewshed analysis," Master's thesis, Dept. Comput. Sci. Eng., Chalmers Univ. Technol., Gothenburg, Sweden, 2015.

[27] Y. Zhao, A. Padmanabhan, and S. Wang, "A parallel computing approach to viewshed analysis of large terrain data using graphics processing units," *Int. J. Geograph. Inf. Sci.*, vol. 27, no. 2, pp. 363–384, 2013.

[28] N. Stojanović and D. Stojanović, "Performance improvement of viewshed analysis using GPU," in *Proc. 11th Int. Conf. Telecommun. Modern Satell., Cable Broadcast. Serv.*, 2013, vol. 2, pp. 397–400.

[29] L. Toma, "Viewsheds on terrains in external memory," *SIGSPATIAL Special*, vol. 4, no. 2, pp. 13–17, 2012.

[30] A. Osterman, L. Benedičič, and P. Ritoša, "An IO-efficient parallel implementation of an R2 viewshed algorithm for large terrain maps on a CUDA GPU," *Int. J. Geograph. Inf. Sci.*, vol. 28, no. 11, pp. 2304–2327, 2014.

[31] X. Song, G. Tang, X. Liu, W. Dou, and F. Li, "Parallel viewshed analysis on a PC cluster system using triple-based irregular partition scheme," *Earth Sci. Informat.*, vol. 9, no. 4, pp. 511–523, 2016.

[32] J. C. C. Bravo, T. Sarjakoski, and J. Westerholm, "Efficient implementation of a fast viewshed algorithm on SIMD architectures," in *Proc. 23rd Euromicro Int. Conf. Parallel, Distrib., and Netw.-Based Process.*, 2015, pp. 199–202.

[33] A. J. Cauchi-Saunders and I. J. Lewis, "GPU enabled XDraw viewshed analysis," *J. Parallel Distrib. Comput.*, vol. 84, pp. 87–93, 2015.

[34] K. Mills, G. Fox, and R. Heimbach, "Implementing an intervisibility analysis model on a parallel computing system," *Comput. Geosci.*, vol. 18, no. 8, pp. 1047–1054, 1992.

[35] M. Zaharia *et al.*, "Apache Spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[36] R. Shyam, B. G. HB, S. Kumar, P. Poornachandran, and K. Soman, "Apache spark a big data analytics platform for smart grid," *Procedia Technol.*, vol. 21, pp. 171–178, 2015.

[37] D. Lunga, J. Gerrand, L. Yang, C. Layton, and R. Stewart, "Apache Spark accelerated deep learning inference for large scale satellite image analytics," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 13, pp. 271–283, Jan. 2020.

[38] J. Sun *et al.*, "An efficient and scalable framework for processing remotely sensed big data in cloud computing environments," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 7, pp. 4294–4308, Jul. 2019.

[39] Z. Wu *et al.*, "Scheduling-guided automatic processing of massive hyperspectral image classification on cloud computing architectures," *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2020.3026673.

[40] E. Johansson and J. Lundberg, "Distributed viewshed analysis an evaluation of distribution frameworks for geospatial information systems," Master's thesis, Dept. Comput. Sci. Eng., Chalmers Univ. Technol., Gothenburg, Sweden, 2016.

[41] Y. Wang and S. Wang, "Research and implementation on spatial data storage and operation based on Hadoop platform," in *Proc. 2nd IITA Int. Conf. Geosci. Remote Sens.*, 2010, vol. 2, pp. 275–278.

[42] S. Ladra, J. R. Paramá, and F. Silva-Coira, "Scalable and queryable compressed storage structure for raster data," *Inf. Syst.*, vol. 72, pp. 179–204, 2017.

[43] D. Borthakur, "The Hadoop Distributed File System: Architecture and design," *Hadoop Project Website*, vol. 11, 2007, Art. no. 21.

[44] J. Zhang, S. Zhou, T. Liang, Y. Li, C. Chen, and H. Xia, "A two-level storage strategy for map-reduce enabled computation of local map algebra," Earth Sci. Informat., vol. 13, pp. 479–492, 2020.

Jianbo Zhang received the Ph.D. degree in cartography and geographical information engineering from the China University of Geosciences, Wuhan, China, in 2006.

He is currently an Associate Professor with the School of Geography Information Engineering, China University of Geosciences. His research interests include high-performance geocomputation and spatiotemporal analytics, and big data mining.

Subin Zhao received the B.E. degree in software engineering from the China University of Geosciences, Wuhan, China, in 2018, where she is currently working toward the master's degree in major of software engineering.

Her current research interests include spatial online analytical processing and big data mining.

Zhuangzhuang Ye received the B.E. degree in information engineering from the China University of Geosciences, Wuhan, China, in 2019, where he is currently working toward the master's degree in major of software engineering.

His current research interest includes geocomputation.