Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 111 (2017) 120–128

Procedia Computer Science

www.elsevier.com/locate/procedia

8th International Conference on Advances in Information Technology, IAIT2016, 19-22 December 2016, Macau, China

Efficient visibility analysis for massive observers

Wentao Wang[a,c], Bo Tang[b], Xiaopeng Fan[c], Haixia Mao[d], Hang Yang[e], Min Zhu[a]*

[a]*Sichuan University, Chengdu 610065, China*
[b]*The Hong Kong Polytechnic University, Hong Kong, China*
[c]*Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China*
[d]*Shenzhen Polytechnic, Shenzhen 518055, China*
[e]*China Southern Power Grid, Guangzhou 510000, China*

Abstract

Many applications in Geographic Information System (GIS) apply visibility analysis as a key subroutine, and thus the time spent on visibility analysis is the bottleneck for all these applications, such as navigation, aviation, landscape, and military etc. The new challenge to the scalability of visibility analysis for large datasets shows, most of academic works in GIS only consider a few thousands of observer objects, while many works in industry and science have to face on millions (even billions) of observer objects. In this paper, we devise a novel computation framework which consists of three components, i.e., optimized line-of-sight algorithm, R*-tree filter and MapReduce-based segmented computation. The proposed solution can support GIS systems to conduct efficient visibility analysis for massive observers. Finally, we demonstrate the efficiency and the scalability of our proposed solutions by synthetic datasets. The results show that our proposed solution achieves at least an order of magnitude speedup over existing solutions.

Peer-review under responsibility of the organizing committee of the 8th International Conference on Advances in Information Technology

Keywords: Visibility analysis; scalability; MapReduce; Regular Square Grids; R*-tree;

* Corresponding author.Tel:+ 18202871861
E-mail address: zhumin@scu.edu.cn

1877-0509
Peer-review under responsibility of the organizing committee of the 8th International Conference on Advances in Information Technology
10.1016/j.procs.2017.06.018

1. Introduction

Visibility analysis is one of the core modules in many GIS applications. It is referred to as determining the visibility among the terrain or other objects for given observer points. For example, give the terrain as Fig. 1 and observer point A, an observer can see the target B from A so that the target point B is visible for A. However, the target point C is invisible as the terrain blocks the sight of an observer from A to C. The problem of visibility analysis is to decide whether there is a straight line (i.e., line of sight) between the observer point and the target point, which is not blocked by some obstacles (e.g., buildings, terrain). In general, the visibility between observer point and target point is symmetry, e.g., in Fig. 1, B is visible from A and A is also visible from B[1]. On the contrary, C is not visible from A and A is not visible from C.

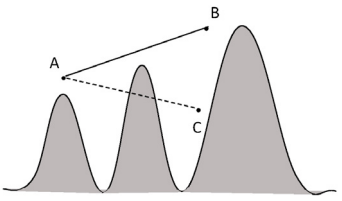

Fig. 1. Visibility analysis illustration.

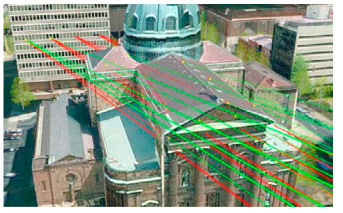

Fig. 2. Visibility analysis of points.

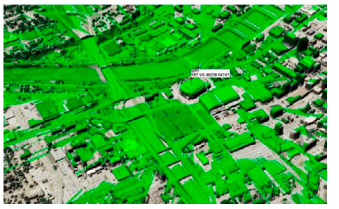

Fig. 3. Viewshed analysis.

Visibility analysis has been widely used in many applications, such as military[2], archaeological[3], public security[4]. In the field of smart city, visibility analysis is not only widely used in landscape or building management[5], but also bring benefits to traffic navigation[6] application.

Visibility analysis is a key subroutine in applications above. Thus the time spent on visibility analysis is the bottleneck of the performance of these applications. However, with more and more users and devices, traditional visibility analysis algorithms degenerates in brute force method. New challenges result from the following reasons. On one hand, users require more accurate and high-resolution analysis results. On the other hand, the devises, the sample points and the observer points increase dramatically, i.e., there are almost one thousand observation points in the same building.

In addition, with the increasing points, traditional in-memory visibility analysis algorithms become unpractical because these data points cannot be loaded and computed in memory at once.

There are two variants of the problem of visibility analysis.

Case I, visibility analysis of points: There are multiple points (i.e., observer and target points), as shown in Fig. 2. The result of visibility analysis between two points can be true (i.e., the observer point can reach target point), or false (i.e., there are some obstacles between two points), so that an observer at the observer point cannot see the target point directly.

Case II, viewshed analysis: Viewshed analysis is referred to as the analysis to discover the visible areas from some specific observation points, as shown in Fig. 3.

The most efficient viewshed analysis algorithm, which implemented on single machine by using GPU[7], is fast and scalable, but the computation cost is still too expensive to be extended for multiple observers, or for online applications especially.

In practice, the cost of visibility analysis is too expensive to compute visibility for each pair of points at single machine. In this paper, we take advantage of the MapReduce model from big data analytics in order to accelerate the process of visibility computation. Therefore, we are able to carry out visibility analysis with massive observer and target points within reasonable time.

The major technical contributions of this paper are listed as follows:

(1) We propose an efficient computation framework to conduct visibility analysis in big data analytics.

(2) We present a suite of optimization techniques (i.e., optimized line-of-sight algorithm, R^*-tree) to improve the efficiency of visibility computation between one observer point and multiple target points.

The rest of this paper is organized as follows. We first review the related work in Section 2, then formulate the problem of visibility analysis and introduce the base-line solution in Section 3. In Section 4, we propose several

optimization techniques to accelerate the base-line solution and accelerate the optimized algorithm by exploiting the advantages of the MapReduce model. The experimental evaluations are illustrated in Section 5. We conclude this work in Section 6.

2. Related work

With the rapid development of sensor techniques, the collection of vast amounts of terrain data with high resolution[8] becomes possible. Thus more and more users request the high accurate results of visibility analysis. However, with the increasing data points collected, the in-memory visibility analysis becomes impossible.

Haverkort et al., proposed an I/O efficient external memory variant of Van Kreveld's algorithm[9]. It computes a tight upper bound for fast visibility grid computation in internal memory, which can be used to in large data sets scenario (i.e. cannot fit in memory). Osterman et al., introduced the parallel viewshed algorithm on the compute unified device architecture (CUDA)[7], which exploits the high parallelism of the graphics processing unit (GPU). This algorithm is the most efficient viewshed approach for large datasets, which can spend about 7s to compute visibility between one observer point and 5×10^9 target points. However, the performance of this GPU-based algorithm degenerates dramatically with the increasing of observer points.

Recently, the combination of the spatial analysis problem with big data computation framework, i.e., Hadoop and Spark, is a hot topic both in industry and research areas. In industry, Esri provides a GIS tool as one of the Hadoop toolkits, which allows users to leverage the Hadoop computation power to conduct spatial analysis on spatial data. On the other hand, there are many literatures about this topic in academia. Aji et al. provided a scalable and high performance spatial data warehousing system (Hadoop-GIS) for running large scale spatial queries on Hadoop[10]. GeoSpark, which was introduced by Yu et al[11], is a memory cluster computing framework for processing large scale spatial data. Unfortunately, none of the above works can address the visibility analysis problem.

3. Efficient visibility analysis solution in single machine

3.1. Problem definition

We first present the definitions of terrain and point. For terrain data, there are two different digital elevation models[12] to represent it in geographic information science: (1) triangulated irregular networks (TINs), and (2) regular square grids (RSGs).

A terrain based on TINs consists of a collection of planar, irregular triangular patches. While a terrain based on RSGs consists of equally sized grid cells which contain elevation information[13]. In this work, we refer the terrain (cf. Definition 1) as Regular Square Grids (RSGs) as it widely used in literature[12].

Definition 1 (Terrain): *A spatial data model that defines space as an array of equally sized cells arranged in rows and columns, which called **T**. Each cell contains an attribute value (i.e., elevation) and location coordinates.*

Without loss of generality, the terrain is modeled by a 3-D space, where width and height are usually limited, i.e., 100×100 m^2, or 500×500 m^2.

Definition 2 (Observer / target point): *A observer / target point is a three dimensions' coordinates in the terrain space, denotes as p: (x, y, z).*

Then, we define visibility analysis problem as follows.

Definition 3 (Visibility Analysis Problem): *Given a terrain **T**, a set of observer points **O**={o_1, o_2, ⋯, o_n} and a set of target points **P**={p_1, p_2, ⋯, p_m}, the visibility analysis return the computation result of isVisible(o_i, p_j, **T**), with $i\in[1,n]$, $j\in[1,m]$.*

For each point pair, i.e., oi and p_j, the result of isVisible(o_i, p_j, ***T***) is TRUE or FALSE, where TRUE means o_i and p_j are visible with terrain ***T***, otherwise the result will be FALSE.

In Chapter 2 of[13], it defined viewshed analysis problem. However, it does not cover all this visibility analysis in our paper. Thus, its solution cannot adapt to our problem directly. Inspired by[13], we convert both viewshed analysis and visibility analysis of points to points visibility analysis problem. Before analysis the visibility between two points, we first define the gradient of point e, with respect to observer point oi as follows.

$$\mathrm{Gradient}(e, o_i) = (e.z - o_i.z) / D(e, o_i)$$

where e.z and o_i.z are elevations of e and oi, D(e, o_i) is the distance between e and o_i which represented as

$$\mathrm{D}(e, o_i) = \sqrt{(e.y - o_i.y)^2 + (e.x - o_i.x)^2}$$

Then, according to[15], the visibility analysis problem can be addressed by the following lemma:

Lemma 1 (Visible Points): *The target point p_j is visible from o_i if and only if the gradient of p_j is larger than the gradient of any point e intersected by the Line Of Sight (LOS) from oi to p_j in the projection of **T**.*

Proof 1. *We omit the details of lemma proof as it similar with that in[13]. However, we can remove the arctan function as it is monotonic.*

3.2. Solution framework

There are many existing algorithms to compute visibility between two points base on Lemma 1. Unfortunately, these existing algorithms always tailored for specified applications. More specifically, most of these algorithms just consists of two phases: initialization and incremental traversal. Thus, in this work, we propose a basic computation framework, which can be applied to different specific scenarios, e.g., digital differential analyzer[14], Amanatides and Woo's algorithm[15].

The basic algorithm to compute the visibility between two points o_i and p_j as shown in Algorithm 1. First, the *Init* function obtains the computation step direction, slop of LOS, which will be used in *Next*. Second, the *Next* function obtains the next intersection point according to afferent point and the result of the *Init* function. If NULL value is returned by *Next*, it means the end of incremental traversal phase. Third, the algorithm compares the gradient of intersection point e with o_i with the gradient of target point p_j with oi iteratively. Finally, if the output is TRUE, the two points oi and p_j are visible. Otherwise, they are invisible, e.g., blocked by terrain.

As mentioned before, the *Init* function corresponds to the initialization phase of the algorithm while the *Next* function corresponds to an iteration of incremental traversal phase. In our work, the *Init* and *Next* function is similar to Amanatides and Woo's algorithm[15].

Until now, we present the basic algorithm to compute the visibility between two points. We then extend this algorithm to process visibility analysis among one observer point with multiple target points (i.e., VisibilityOfMultiPoints). It is trivial because it calls VisibilityOfOnePoint several times for each observer and target point pair.

Complexity Analysis: Let ***T*** be a terrain represented as a grid of m square cells. Computing visibility between one observer point and n target points requires $O(n\sqrt{m})$ time by using VisibilityOfMultiPoints.

Algorithm 1 Visibility between Two Points

```
1: function VisibilityOfOnePoint(o_i, p_j, T)
2:     Init(o_i, p_j)
3:     grad ← Gradient(p_j, o_i)
4:     e ←Next(o_i, p_j, T)
5:     while e! = NULL do
6:         if Gradient(e, o_i) > grad then
7:             return FALSE
8:         e ←Next(e, p_j, T)
9:     return TRUE
```

Algorithm 2 Visibility between Two Points

```
1: function VisibilityOfOnePoint'(o_i, center, T)
2:     Init(o_i, center)
3:     maxGrad ← −INFINITE
4:     e ←Next(o_i, center, T)
5:     while e! = NULL do
6:         grad ← Gradient(e, o_i)
7:         if maxGrad < grad then
8:             maxGrad ← grad
9:         e ←Next(e, center, T)
10:    return maxGrad
```

3.3. Optimization techniques

In this section, we proposed several optimization techniques upon the basic computation framework. More specifically, we focus on how to exploit the *Init* and *Next* functions to accelerate visibility analysis directly.

In practice, the target points in smart city area are sampled from landscapes and sunshine area. For points from same landscape, the differences between their coordinates (x and y) are always relatively small which means many of them are projected in the same grid cell on ***T***.

Base on above observation, it is not necessary to compute visibility for each target point one by one. We use center point of a grid cell to represent these points which mapped into the cell. For the center point of each cell, we just compute the "minimum height" that meets the demand of asserting points over it are visible. For each real point projected in that cell, we only need to compare its height to "minimum height" to determine visibility.

We modified above VisibilityOfOnePoint as follows: (1) the real target point is not a parameter and replaced by the center point of grid cell; (2) in each iteration, we compare the maximum gradient among intersection points. (3) the output is maximum gradient between intersection points. The improved VisibilityOfOnePoint is shown as Algorithm 2.

Thus, for one observer with multiple single point, we employ a hash map to store the center points of cells and the corresponding "minimum height". For each target point, we first obtain its center point of the cell into which it maps. We then check whether hash map contains this center point. If it contains that point, we can use the corresponding "minimum height" to determine this point's visibility directly. Otherwise, we use VisibilityOfOnePoint' to compute visibility and then insert the center point and the calculated "minimum height" into hash map.

Complexity Analysis: After we introduce above optimization technique, the computation cost of VisibilityOfMultiPoints' is $O(n+m\sqrt{m})$. It is more efficient than the above VisibilityOfMultiPoints, whose time complexity is $O(n\sqrt{m})$.

3.4. R-tree: A solution for multiple observers*

In this section, we aim to optimize computing visibility between multiple observer points and multiple target points, while the goal is to reduce time cost for computing visibility of multiple target points from the same observer point. An intuitive way to compute visibility of multiple observer points is just calling VisibilityOfMultiPoints or VisibilityOfMultiPoints' for each observer point. But such a method always brings us many unnecessary computations.

Before introducing our optimization scheme, we first give two definitions as follows:

Definition 4 (Box): *A virtual three-dimensional rectangle in space which known as bounding boxes or minimum bounding rectangles.*

Definition 5 (Visible State of Box): *The visible state of points in the box from a given observer point. If all points in a box is visible (or invisible) from the observer point, the box is visible (or invisible) from the observer point. Otherwise, the box is partial visible.*

To reduce the cost of computing time, we take the following method. First, we use R*-tree to partition target points into many boxes. Before computing the visibility of points in a box, we calculate visible state of box using few sampling points on borders of the box and then determine whether to compute visibility of points in box according to the visible state of box. This process reduced many unqualified computations.

The reason why we can determine visible state of a box by using few sample points on its borders is that the terrain information is represented by RSGs. As long as the gap between two adjacent sample points is less than the cell size, we can determine visible state of the box by these sample points.

Next, we will describe how to determine visible state of a box from an observer point through a small amount of sample points on borders of the box.

Box is invisible. To determine a box is not visible, we just need to calculate visibility of these sample points and if all of these sample points are not visible, the box is considered not visible.

Box is visible. Determining a box is visible is more complicated. There are two cases as follows:

(1) The height of the observer point is smaller than minimum height of the box, we can assert the box is visible if all of sample points are visible.

(2) The height of observer point is larger than minimum height of the box, we cannot determine the box's visible state just using the sample points. The reason is that there may be some obstacles under sight lines of the observer point and the sample points by accident. Under this case, to assert the box is visible, we must ensure minimum height of the box is larger than maximum height of obstacles which is under the box as well as all of sample points are visible.

Box is partial visible. When visible states of these sample points are not the same, it means the box is partial visible.

If the box is visible (or invisible) from an observer point, it indicates all points in the box are visible (or invisible) from the observer point. Only when the box is partial visible, we have to compute visibility between the observer point and every target point in the box.

This method is always using to compute visibility between multiple observer points and multiple target points. Because we only need to build a R^*-tree once for these target points, but can use the R^*-tree to filter unnecessary computations for each observer point. It is not suggested to use R^*-tree when the number of observer points is relatively small. The main reason is the R^*-tree construction time will dominate total cost instead of actual visibility analysis cost.

4. Efficient visibility analysis for massive observers

In Section 3, we proposed a feasible solution to the visibility analysis problem. In this section, we propose to extend above visibility analysis computation framework to big data analytics (i.e., millions even trillions of observer objects) in industry.

It is non-trivial to extend the visibility analysis computation framework to big data platform (i.e., Hadoop) as the visibility analysis framework consists of three basic components: (1) terrain, (2) observer points set and (3) target points set.

We propose to exploit the advantages of big data platform (i.e., Hadoop) to conduct visibility analysis. The challenges are: (1) How to exploit MapReduce model in Hadoop to accelerate visibility analysis? and (2) How to synchronize these three basic components in the distributed computation environment.

In order to address the above challenges, we proposed a MapReduce-based visibility analysis computation framework as follows, as shown in Fig. 4:

1) Divide all observer points into blocks, and assign each block to a map task.

2) Obtain all possible target points and terrain data within the maximum visual range of observer points for each block in map tasks.

3) Compute visibility for these observer points and target points by using optimized algorithm in Section 3.3 and Section 3.4.

4) Reduce the computation result and output all these results.

This MapReduce-based computation framework has following advantages: (a) Load the computation resources into memory at once; (b) Compute as little as possible observer and target pair in each task; (c) Incorporate optimization techniques to each computation unit (i.e., map-reduce tasks).

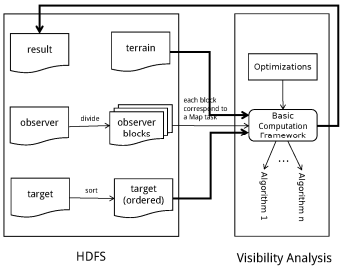

Fig. 4. Overall flow of computing visibility on Hadoop.

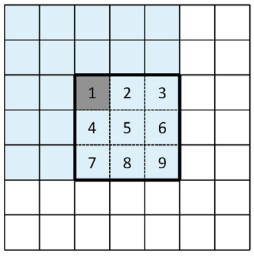

Fig. 5. The illustration of partition observer points.

There are three core functions: divide, sort and compute as follows:

Divide. We project all observer points into the grid cells of $\boldsymbol{T}$ and divide them according to these cells. More specifically, we first divide all cells into two parts: middle part and edge part. The middle part, shown as the black-bordered part in Fig. 5, is mainly used to generate block through a fixed size, named block size. The edge part, whose length is always equal to the maximum visual distance, consists of the cells not in the middle part. Finally,

observer points will be projected into these cells. For point mapped into the middle part, we assign id of the block into which it projects to the point's id value, while for point mapped into edge part, we update its id value with id of the block whose distance from it is smallest.

After we divide all these observer points into blocks, we then greedy search the related terrain and target points for them.

For the terrain data, we fetch the subset which we are interested from its ordered storage directly. However, for target points, we should scan all the data to obtain the corresponding subset. An intuitive solution is partitioning all target points into blocks by the maximum visual range of an observer block. For example, we can regard all the target points in blue region as one block in Fig. 4. It is simple but ineffective as many of target points will duplicate in different blocks. In addition, it is not space efficient solution. We propose sort-based solution as follows:

Sort. We first project all target points into grid cells of ***T***. Then we sort the target points by row and column values of the cells. For the points projected into same grid cell, we don't care about their internal order.

Finally, we define the compute function.

Compute. In this phase, we compute visibility on each worker for its corresponding block of observer points. In the setup function, we load terrain and target points data from HDFS into memory according to current block id and build up a R^*-tree for the target points. In the map function, we only compute visibility by using the optimized algorithm in Section 3 for each computing task.

5. Experiments

We evaluate the proposed techniques in this Section. There are four experiments in this section. We first evaluate the accuracy and efficiency of our optimization algorithms in Section 3. We then demonstrate the scalability of visibility analysis on big data scenario in Section 4.

We implemented all visibility analysis algorithms, including VisibilityOfMultiPoints (VOMP) and VisibilityOfMultiPoints' (VOMP'), in Java.

Datasets. All the datasets in these experiments are generated by following the distribution of a real industry dataset. The terrain data are in raw format. Table 1 shows the detail information of terrains used in following experiments. Both observer and target points are use double float precision with 3-D coordinates.

Platform. Our system consists of a cluster with 9 virtual machines. Each virtual machine is with triple-core 2.4GHz processor, 4096KB L2 cache per processor, 8 GB RAM, and a Seagate SATA 7200-RPM hard drive. The algorithm RAM limitation in Hadoop is 4 GB.

Experiment 1. To evaluate the accuracy of our optimization visibility analysis algorithms, we computed visibility of 5×10^5 target points from a given observer point on 500×500 terrain (terrain 1). And the accuracy is calculated through comparing to the Line-of-Sight module in ArcGIS. We assume that all the results from ArcGIS is right. VOMP'+R^*-tree means to we use VOMP' and R^*-tree simultaneously.

As Table 2 shows, although the basic visibility analysis algorithm is simple, it works perfectly. The accuracy of VOMP is almost the same with the Line-of-Sight module in ArcGIS. The accuracy of VOMP' is lower than VOMP slightly. VOMP'+R^*-tree's accuracy is the same with VOMP', which demonstrates the effectiveness of all optimization techniques about R^*-tree.

Table 1. The information of terrains.

Data set	Grid-cell size	Columns	Rows	Grid cells
1	1×1 m^2	500	500	250000
2	1×1 m^2	1000	1000	1000000
3	1×1 m^2	1500	1500	2250000
4	1×1 m^2	2000	2000	4000000
5	1×1 m^2	2500	2500	6250000
6	1×1 m^2	3000	3000	9000000

Table 2. Accuracy of optimization visibility analysis algorithms.

	VOMP	VOMP’	VOMP’+R^*-tree
Accuracy	0.9902	0.987	0.987

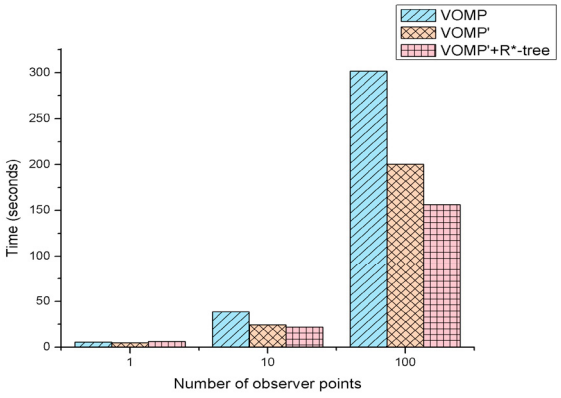

Fig. 6. The computing time comparison for different number of observers.

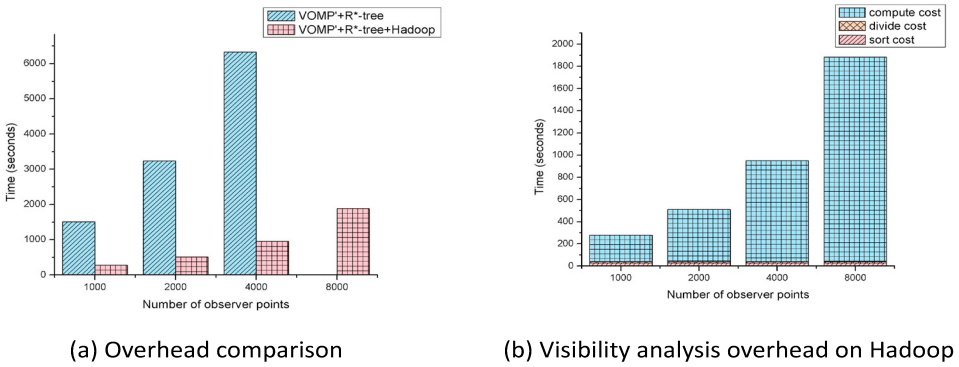

Fig. 7. Overhead analysis.

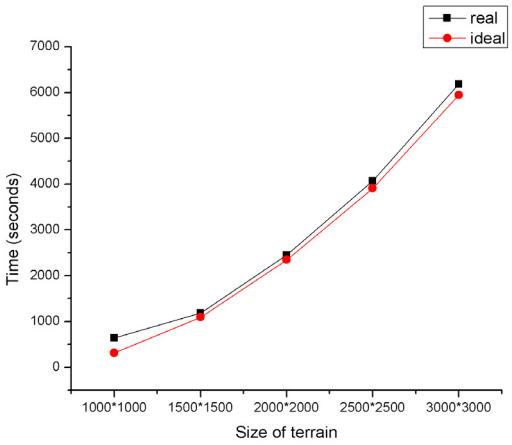

Fig. 8. Scalability analysis.

Experiment 2. In order to show the efficiency of our optimization techniques, we compare the time of computing visibility of 5×10^5 on terrain 1 by varying the number of observer points.

In Fig. 6, VOMP' is more efficient than VOMP no matter how many observer points are. VOMP'+R*-tree spends more time to compute visibility than the other two with only one observer, which is consistent with our analysis, i.e.,

the overhead of constructing a R*-tree will dominate the total cost under such a case. However, with the increment of the observer points, VOMP'+R*-tree performs much better than the others.

Experiment 3. In Fig. 7 (a), the efficiency of VOMP'+R*-tree on Hadoop is much better than VOMP'+R*-tree on single machine. For 8000 observer points, it only spends about half an hour to complete computation on Hadoop, while it is impractical on single machine.

Fig. 7 (b) shows the performance of visibility analysis on the MapReduce-based framework. The cyan bars are the cost of computing visibility, the yellow bars are the cost of dividing observer points, and magenta bars represent the cost of sorting target points. As the figure shows, the costs of partitioning and sorting are very small. The overall cost of visibility analysis in Hadoop is dominated by compute phase.

Experiment 4. In this experiment, we set 200 cells as maximum visual distance and 100 cells as a block size. It means 6×6 blocks for 1000×1000 terrain and 11× 11 blocks for 1500×1500 terrain. There are 2 Map tasks in every node.

As Fig. 8 shows, the red line is the ideal time cost while the black line indicates the real cost. The real cost shares the same trend of ideal cost. Thus, the scalability of MapReduce-based computation framework is stable in large datasets.

6. Conclusion and future work

In this work, we revisit a core subroutine (i.e., visibility analysis) in Geographic Information System. We firstly propose a comprehensive computation framework for visibility analysis, then optimize the performance by a suite of optimization techniques (i.e., optimized LOS algorithm, R*-tree). Then we exploit the advantages of MapReduce in Hadoop and design a MapReduce-based visibility analysis framework for real applications in industry. Finally, we demonstrate the effectiveness and efficiency of our proposed computation frameworks. It achieves at least an order of magnitude speedup over existing solutions. In future, we plan to incorporate our techniques to real GIS system (i.e., ArcGIS) for industry applications.

References

1. Pfister-Altschul E. Comparison of Ground-to-Air Visibility Analysis Methods [J]. 2014.
2. VanHorn J E, Mosurinjohn N A. Urban 3D GIS modeling of terrorism sniper hazards[J]. Social Science Computer Review, 2010.
3. Wheatley D. Cumulative viewshed analysis: a GIS-based method for investigating intervisibility, and its archaeological application[J]. Archaeology and GIS: A European Perspective. London: Routledge, 1995: 171-86.
4. Murray A T, Kim K, Davis J W, et al. Coverage optimization to support security monitoring[J]. Computers, Environment and Urban Systems, 2007, 31(2): 133-147
5. Chamberlain B C, Meitner M J. A route-based visibility analysis for landscape management[J]. Landscape and Urban Planning, 2013, 111:13-24.
6. Iida H, Hiroi K, Kaji K, et al. A proposal of IndoorGML extended data model for pedestrian-oriented voice navigation system[C]//Proceedings of the Seventh ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness. ACM, 2015: 2.
7. Osterman A, BenedičičL, Ritoša P. An IO-efficient parallel implementation of an R2 viewshed algorithm for large terrain maps on a CUDA GPU[J]. International Journal of Geographical Information Science, 2014, 28(11): 2304-2327.
8. Toma L. Viewsheds on terrains in external memory[J]. SIGSPATIAL Special, 2012, 4(2): 13-17.
9. Haverkort H, Toma L, Zhuang Y. Computing visibility on terrains in external memory[J]. Journal of Experimental Algorithmics (JEA), 2009, 13: 5
10. Aji A, Wang F, Vo H, et al. Hadoop GIS: a high performance spatial data warehousing system over mapreduce[J]. Proceedings of the VLDB Endowment, 2013, 6(11): 1009-1020.
11. Yu J, Wu J, Sarwat M. Geospark: A cluster computing framework for processing large-scale spatial data[C]//Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2015: 70.
12. Floriani L, Magillo P. Algorithms for visibility computation on terrains: a survey[J]. Environment and Planning B: Planning and Design, 2003, 30(5): 709-728.
13. Zhao Y, Padmanabhan A, Wang S. A parallel computing approach to viewshed analysis of large terrain data using graphics processing units[J]. International Journal of Geographical Information Science, 2013, 27(2): 363-384.
14. Mayorov F V. Electronic digital integrating computers: digital differential analyzers[M]. Iliffe Books, 1964
15. Amanatides J, Woo A. A fast voxel traversal algorithm for ray tracing[C]//Eurographics. 1987, 87(3): 3-10.