A MapReduce Algorithm for Polygon Retrieval in Geospatial Analysis

Qiulei Guo   Balaji Palanisamy   Hassan A. Karimi

School of Information Sciences, University of Pittsburgh
{qiulei, bpalan, hkarimi }@pitt.edu

***Abstract*—The proliferation of data acquisition devices like 3D laser scanners had led to the burst of large-scale spatial terrain data which imposes many challenges to spatial data analysis and computation. With the advent of several emerging cloud technologies, a natural and cost-effective approach to managing such large-scale data is to store and process such datasets in a publicly hosted cloud service using modern distributed computing paradigms such as MapReduce. For several key spatial data analysis and computation problems, polygon retrieval is a fundamental operation which is often computed under real-time constraints. However, existing sequential algorithms fail to meet this demand effectively given that terrain data in recent years have witnessed an unprecedented growth in both volume and rate. In this work, we present a MapReduce-based parallel polygon retrieval algorithm which aims at minimizing the IO and CPU loads of the map and reduce tasks during spatial data processing. Our proposed algorithm first hierarchically indexes the spatial terrain data using a quad-tree index, with the help of which, a significant amount of data is filtered out in the pre-processing stage based on the query object. In addition, a prefix tree based on the quad-tree index is built to query the relationship between the terrain data and query area in real time which leads to significant savings in both I/O load and CPU time. The performance of the proposed techniques is evaluated in a Hadoop cluster and the results demonstrate that the proposed techniques are scalable and lead to more than 35% reduction in execution time of the polygon retrieval operation over existing distributed algorithms.**

I. Introduction

The proliferation of cost-effective data acquisition devices like 3D laser scanners has enabled the acquisition of massive amounts of terrain data at an ever-growing volume and rate. With the advent of several emerging collaborative cloud technologies, a natural and cost-effective approach to managing such large-scale data is to store and share such datasets in a publicly hosted cloud service and process the data within the cloud itself using modern distributed computing paradigms such as MapReduce. Examples of applications that process such terrain data include urban environment visualization, shadow analysis, visibility computation, and flood simulation. Many geo-spatial queries on such large datasets are intrinsically complex to solve and are often computed under real-time constraints, thus requiring fast response times for the queries. However, most existing sequential algorithms fail to meet this demand effectively given that terrain data in the recent years have witnessed an unprecedented growth in both volume and rate. Therefore, a common approach to speed up spatial query processing is parallelizing the individual operations on a cluster of commodity servers.

Polygon retrieval is a fundamental geospatial operation which is often computed under real-time constraints. Polygon retrieval involves retrieval of all terrain data within a given polygon's boundary [4], [5] to access the spatial data within a specific area of interest for further analysis. We note that terrain data is usually represented using one of the common data structures to approximate surface, for example, digital elevation model (DEM) and triangulated irregular network(TIN). Among these existing structures, TIN[6] is a widely used model and it consists of irregularly distributed nodes and lines arranged in a network of non-overlapping triangles. Compared to other spatial data structures, TIN requires considerably higher storage as it can be used to represent surfaces with much higher resolution and detail. For instance, a TIN dataset for the city of Pittsburgh would require up to 60GB of storage and for the state of Pennsylvania would require up to 60TB. Therefore, real-time processing of such a large amount of data is not possible through sequential computations and a distributed parallel computation is needed to meet the fast response time requirements.

We argue that such large scale spatial datasets can effectively leverage the MapReduce programming model[2] to compute spatial operations in parallel. In doing so, key challenges include how to organize, partition and distribute a large scale spatial dataset across 10s of 100s of nodes in a cloud data center so that applications can query and analyze the data quickly and cost-effectively. Furthermore, polygon retrieval is a CPU-intensive operation whose performance heavily depends on the computation load causing performance bottlenecks when dealing with very large datasets. Therefore, a suitable algorithm needs to minimize the computation load on the individual map and reduce tasks as well. In this paper, we develop a MapReduce-based parallel algorithm for distributed processing of polygon retrieval operation in Hadoop [3]. Our proposed algorithm first hierarchically indexes the spatial terrain data using a quad-tree index, with the help of which, a significant amount of data is filtered out in the pre-processing stage based on the query object. In addition, a prefix tree based on the quad-tree index is built to query the relationship between the terrain data and query area in real time which leads to significant savings in both I/O load and CPU time. We evaluate the performance of the proposed algorithm through experiments on our Hadoop cluster consisting of 20 virtual machines. Our experiment results show that the proposed algorithm is scalable and performs faster than existing distributed algorithms.

The rest of the paper is organized as follows: Section 2 reviews the related work and in Section 3, we provide a background on TIN and overview the polygon retrieval problem. Section 4 describes our proposed MapReduce based polygon retrieval algorithm and the optimization techniques. We discuss the experiment results in Section 5 and conclude in Section 6.

II. Related Work

Polygon retrieval is a common operation for a diverse number of spatial queries in many GIS applications. Willard et. al [4] proposed the polygon retrieval problem formally and devised an algorithm that runs in $O(N^{log_6^4})$ time in the worst-case. To speed up this query further, several efficient sequential algorithms have been proposed. The most notable among these include the algorithms presented in [5], [12], [14], [15]. However, with the recent massive growth in terrain data, these sequential algorithms fail to meet the demands of real-time processing.

As cloud computing has emerged to be a cost-effective and promising solution for both compute and data intensive problems, a natural approach to ensure real-time processing guarantees is to process such spatial queries in parallel effectively leveraging modern cloud computing technologies. In this context, some earlier work [16] had explored the feasibility of using Google App Engine, the cloud computing technology by Google, to process TIN data. Since MapReduce/Hadoop has become the defacto standard for distributed computation on a massive scale, some recent works have developed several MapReduce-based algorithms for GIS problems. The authors in [17] propose and implement a MapReduce algorithm for distributed polygon overlay computation in Hadoop. The authors in [18] present a MapReduce-based approach that construct inverted grid index and processes kNN query over large spatial data sets. The technique presented in [19] creates a unique spatial index and Voronoi diagram for given points in 2D space and enables efficient processing of a wide range of geospatial queries such as RNN, MaxRNN and kNN with the MapReduce programming model. Hadoop-GIS[20] and Spatial-Hadoop[21], [10] are two scalable and high-performance spatial data processing system for running large-scale spatial queries in Hadoop. These systems provide support for some fundamental spatial queries including the minimal bounding box query. However, they do not directly support polygon retrieval operation addressed in our work. In our work, we primarily focus on the polygon retrieval queries on spatial data and we devise specific optimization techniques for an efficient implementation of the parallel polygon retrieval operation in MapReduce.

III. Background

In this section, we provide the required background and preliminaries about the TIN spatial data storage format and a brief overview of MapReduce based parallel processing of large-scale datasets.

A. TIN Data

TIN[6] is a commonly used model for representing spatial data and it consists of irregularly distributed nodes and lines arranged in a network of non-overlapping triangles. TIN data typically gets generated from raster data such as LIDAR (Light Detection and Ranging) which is a remote sensing method that uses light in the form of a pulsed laser to measure ranges to the Earth surface. These light pulsescombined with other data recorded by the airborne system generate precise, three-dimensional information about the shape and surface characteristics. In our work, we consider TIN data generated from LIDAR data using the Delaunay triangulation algorithm implemented by the LASTool [30]. An example of LIDAR data and its corresponding TIN representation is shown in Figure 1(a) and Figure 1(b) respectively.

When it comes to data representation, TINs are traditionally stored as a file, in ASCII or the ESRI TIN dataset file format. To improve the efficiency of processing large TIN datasets, [22], [23] have proposed new TIN data structures and operations for spatial databases that allow storing, querying and reconstructing TINs more efficiently. However, we note that there are no standards on the data structures and operations for TIN [16]; Oracle has defined a proprietary data type and operations for managing large TINs in their own spatial database [24]. In our work, we adopt the data format from [16] which comprises of two types of data entities: TIN_Points and TIN Triangles, as shown in Figure 2. Both types have their unique IDs. The TIN_Points type has five properties and the TIN_Triangles entity has three properties. For the TIN_Point, the Adj_TriangleID[] array stores the IDs of its adjacent triangles. For the TIN_Triangle, the Point_ID array and Coordinate array contain the IDs and coordinates for the three vertices of each triangle.

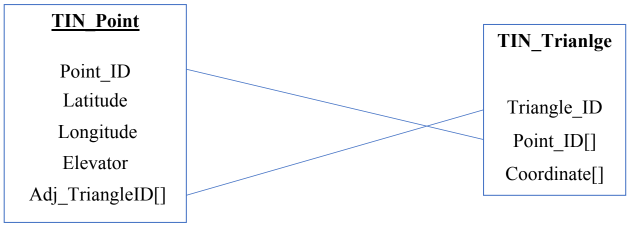
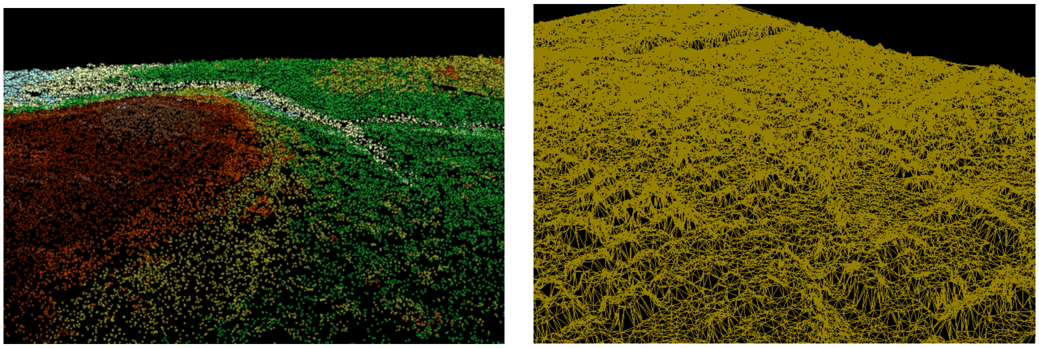


Fig. 2: TIN representation

B. Polygon Retrieval

In this subsection, we describe the polygon retrieval problem using data represented in TIN. Given the boundary of a simple polygon, the polygon retrieval operation retrieves all the terrain data, represented by TIN that intersects with the polygon. As there could be many possible situations of intersection [25], for the purpose of speed, in this work, we consider an intersection when at least one of its vertex of the TIN triangles intersects with the query area. We note that point-in-polygon algorithms can be used to determine whether a point is inside or outside the polygon. One such well-known algorithm is ray tracing algorithm which is usually referred to as crossing number algorithm or even-odd rule algorithm [23]


(a) LIDAR Point cloud

(b) TIN surface

Fig. 1: LIDAR and TIN surface

in the literature.

C. MapReduce overview

In this work, we are focused on MapReduce-based parallel processing of TIN for the polygon retrieval operation. We note that in addition to the programming model, MapReduce [2] also includes the system support for processing the MapReduce jobs in parallel in a large scale cluster. Apache Hadoop[3] is a popular open source implementation of the MapReduce framework. Hadoop is composed of two major parts: storage model, Hadoop Distributed File System (HDFS) and compute model (MapReduce). A key feature of the MapReduce framework is that it can distribute a large job into several independent map and reduce tasks over several nodes of a large data center and process them in parallel.

MapReduce can effectively leverage data locality and processing on or near the storage nodes and result in faster execution of the jobs. The framework consists of one master node and a set of slave nodes. In the map phase, the master node schedules and distributes the individual map tasks to the worker nodes. A map task executing in a worker node processes the smaller chunk of the file stored in HDFS and passes the intermediate results to the appropriate reduce tasks executing in a set of worker nodes. The reduce tasks collect the intermediate results from the map tasks and combine/reduce them to form the final output. Since each map operation is independent of the others, all maps can be performed in parallel. It is also the same with reducers as each reducer works on a mutually exclusive set of intermediate results produced by mappers.

IV. MapReduce-based Parallel Polygon Retrieval

In this section, we first present a naive implementation of parallel polygon retrieval operation using MapReduce and illustrate its performance. We then present our proposed optimization techniques that significantly improves this basic polygon retrieval algorithm.

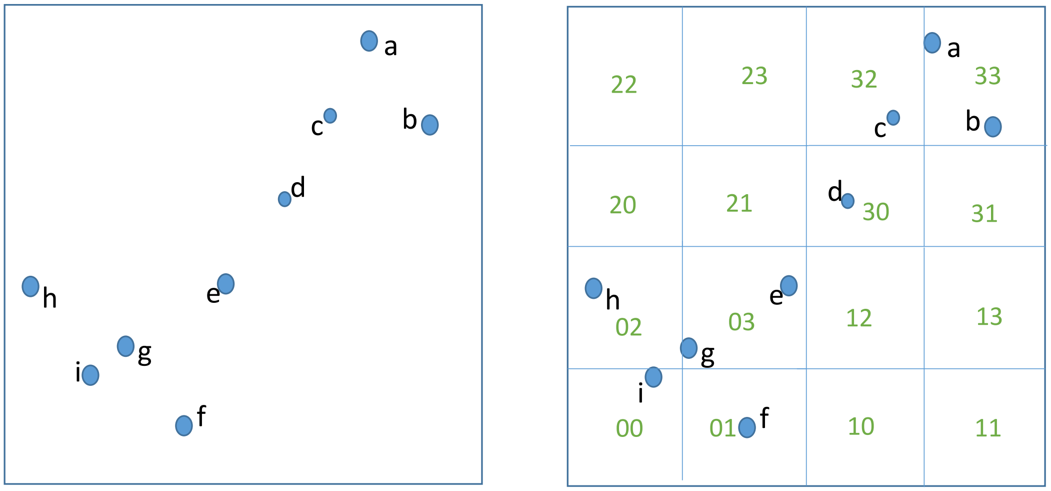
A. Basic MapReduce Algorithm for Polygon Retrieval

An intuitive and straight-forward MapReduce-based polygon retrieval implementation is to process all the terrain data stored in HDFS as part of the MapReduce job. Each mapper will process an input split and check whether a given point is within the boundary of the query area or not. The HDFS partitions the TIN data into several chunks (64 MB blocks by default) and each map task would process one chunk of data in parallel. Unfortunately this basic MapReduce algorithm (Algorithm 10) has several key performance limitations. Firstly, for each query the algorithm reads all terrain data from the HDFS and processes them in the map phase. This approach is not efficient in situations when the query area is a smaller portion of the whole dataset, where the system does not need to scan all terrain data to obtain accurate results. We also note that the point in polygon computation in the map phase is a reasonably CPU consuming operation and hence performing this computation for a huge amount of data will result in significantly longer job execution times.

Our proposed algorithm employs a sequence of optimization techniques that overcome the above-mentioned shortcomings. First, our proposed technique divides the whole dataset stored in HDFS into several chunks of files based on a quad-tree prefix. Then for each range query, we use a prefix tree to organize the set of quad-indices whose corresponding grids intersect the query area. Prior to processing a query, we employ these indices to filter the unnecessary TIN data as part of the data filtering stage so that unwanted data processing is minimized in the map phase. Finally, the proposed approach pre-tests the relationship between the TIN data and the query shape through the built prefix tree in the map function in order to minimize the computation. We describe the essence and details of these techniques in the following subsections.

Algorithm 1 Basic MapReduce Polygon Retrieval

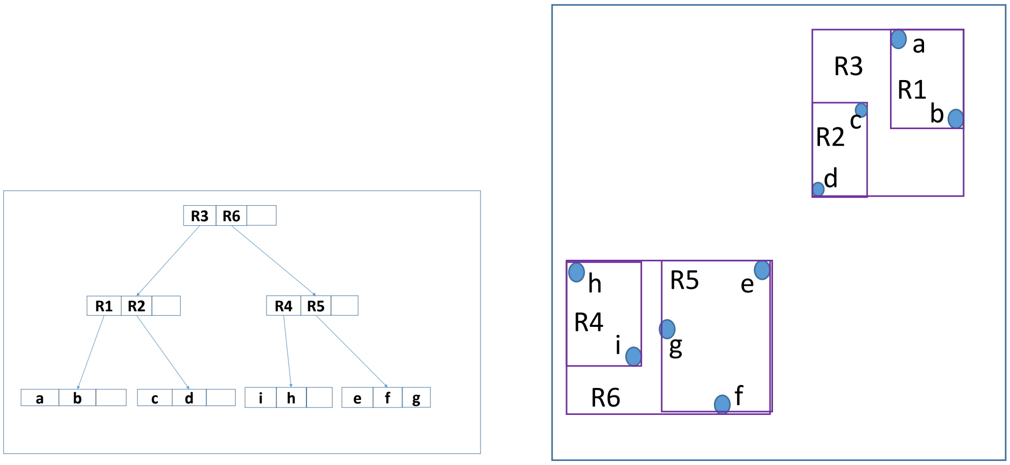
```
$point\_id$: a point ID
$TIN\_point$: a TIN point in space
procedure MAP($point\_id, TIN\_point$)
    get the boundary of the query area from the global
    share memory of Hadoop
    if $tin\_point$ is within the boundary then
        emit the key-value pair ($point\_id, TIN\_point$)
    else
        return
    end if
end procedure
```



(a) Spatial area

(b) Quad-index partitioning

Fig. 3: Spatial Grid Index



(a) R-tree

(b) R-tree partitioning

Fig. 4: Example: R-tree representation

B. Indexing Terrain Data using Quad-tree

To accelerate the processing of terrain data, we divide the entire space based on a quad-tree [13] and index each TIN record using the quad-tree. Quad-tree is a common tree data structure used in many geospatial applications which partitions a two-dimensional space by recursively subdividing the space into four equal regions. Compared with other spatial indices such as R-tree or uniform grid index, the quad-tree has several advantages for polygon retrieval. For instance, unlike the index of R-tree which needs to be built by the insertion of the terrain data one after another while maintaining the tree's balance structure which takes $O(n*logn)$ time, quad- tree index does not need to maintain a real tree and can be used to partition the space directly as shown in Fig 3(b). In addition, with the quad-tree index, we can even infer the topological relationship of the terrain data and the query area from the indices' prefix directly.

An example that compares the index of R-tree and quad-tree is illustrated in Figures 3 and 4. The original space is shown in Figure 3(a). The circles represent the elements of a terrain like the TIN points. Figure 3(b) shows the partitioning of the whole space based on the quad-tree index of length 2 directly. In Figure 4(a), we find a R-tree representation of the same space by going over the elements in the terrain and inserting them to the tree for the spatial partition shown in Figure 4(b). Next, we show how the quad-tree index helps infer the topological relationships using the example shown in Figure 3(b). In the figure, we find the query area shown as a gray region and the set of grid indices that intersect with this gray query area are {*30, 31, 32, 33, 23, 21, 12, 13,03*}. A key observation here is that if a grid cell is within a query area, then all its sub grids are also guaranteed to be within the query area. Therefore the grids' set {30, 31, 32, 33 } can be combined into a single grid cell {*3*} and the indices set in Figure 3(b) can be abbreviated to {*3, 23, 21, 12, 13, 03* }. Thus, if the prefix of a spatial object's quad-index exists in a set, then the object is guaranteed to be within the query area. This property of the proposed indexing scheme avoids the point-in-polygon computation in the map phase enabling the MapReduce jobs to complete significantly faster.

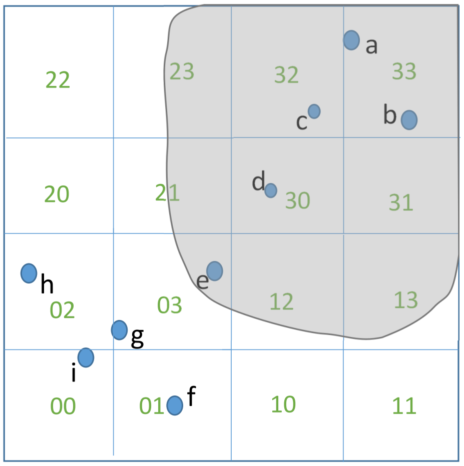


Fig. 5: Polygon Retrieval

To decide whether the prefix of a quad-tree exists in a given set of index entries, it would cost $O(k*n)$ time for a straightforward algorithm, where k is the length of the quad-tree index (also the depth of the quad-tree) and n is the number of entries the indices set. Thus, when n and k get larger, the cost of the algorithm will grow significantly. In the next subsection, we propose a prefix tree structure organizing grid entries that interact with the query area with the cost of only $O(k)$ time.

C. Organizing the index using Prefix Tree

A prefix tree, also called radix tree or trie, is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings [11]. The idea behind a prefix tree is that all strings that share a common prefix inherit a common node. Thus with our prefix tree optimization, testing a prefix of a quad-tree index in a given set can be accomplished in just $O(k)$ time. Figure 7 shows the prefix tree built from the grid indices set {*3,23, 21, 12, 13,03* } of the previous example.

Next, we discuss in detail the optimized query processing algorithm that minimizes the point-in-polygon computation by building a prefix tree based on the grid index and their intersection with the query area. In the pre-processing stage, we first consider the first four grid cells and recursively test them to find overlap with the query area. When a grid cell intersects the query area, we subdivide the grid cell into four sub-cells recursively unless we are at the deepest level of the quad-tree. If the grid cell is within the query area, we stop subdividing the grid cell and insert its index into the prefix tree and mark the corresponding leaf node as ”*w*ithin”. As pointed out earlier, if the grid is within the query area, all its sub grids will also be within the query area and hence this leaf node will always be a leaf node. From the perspective of prefix tree, if the prefix of a quad-tree ends in a leaf node, it means that the corresponding TIN records are within the query area. Finally if a grid cell is outside the query area, we simply ignore it. Here we note that unlike the traditional strategies that subdivide the grid cells based on how many elements are within each grid, we subdivide each grid based on their relation with the query area which significantly improves the performance of our range query processing. We present a complete pseudo code of this algorithm in Algorithm 17.

Algorithm 2 Prefix tree construction

```
$depth$: depth of the prefix tree
$query\_shape$: the geometric shape of the given query
$grid\_queue$ : a queue containing the grid index entries
$ptree$: the output prefix tree
**procedure** BUILDPREFIXTREE($depth, query\_shape$)
    $grid\_queue = \{0, 1, 2, 3\}$
    **while** $grid\_queue.empty() == false$ **do**
        $g = grid\_queue.pop()$
        **if** $query\_shape$ contains $g$ or $g.length() == depth$ **then**
            insert the index $g$ into the prefix tree $ptree$
            mark the leaf node as *within* or *overlapping* the query area
        **else**
            Insert the four child nodes of $g$ into $grid\_queue$
        **end if**
    **end while**
    return ptree;
**end procedure**
```

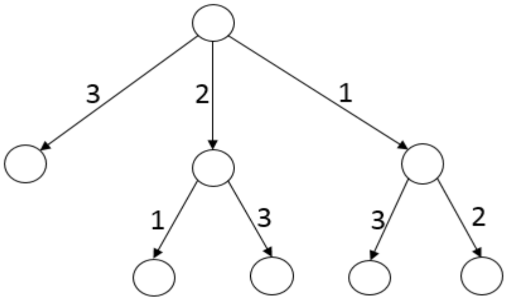
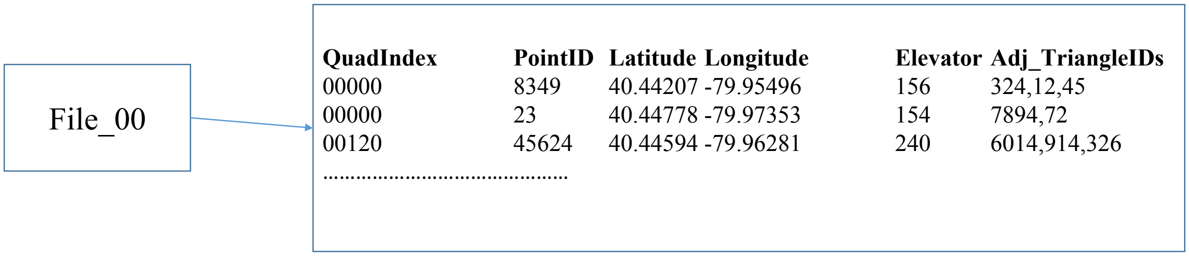


Fig. 7: Prefix Tree

After the prefix tree is created in the pre-processing stage, it is effectively used in the map function. When each mapper receives a TIN record, the relation of the TIN record and the query area is inferred based on the prefix tree created in the pre-processing phase. We first try to search the longest prefix of the TIN record's quad-tree index in the prefix tree which ends in a leaf node. If the search returns nothing, it means that the TIN record is totally outside the query area. If returned tree node is marked as ”*w*ithin”, we will output the TIN record directly. Only if not, we need to do the point-in-polygon computation. Thus, the point-in-polygon computation is avoided for a majority of cases making the algorithm very efficient and scalable. We show a pseudo code of this procedures in Algorithm 17.

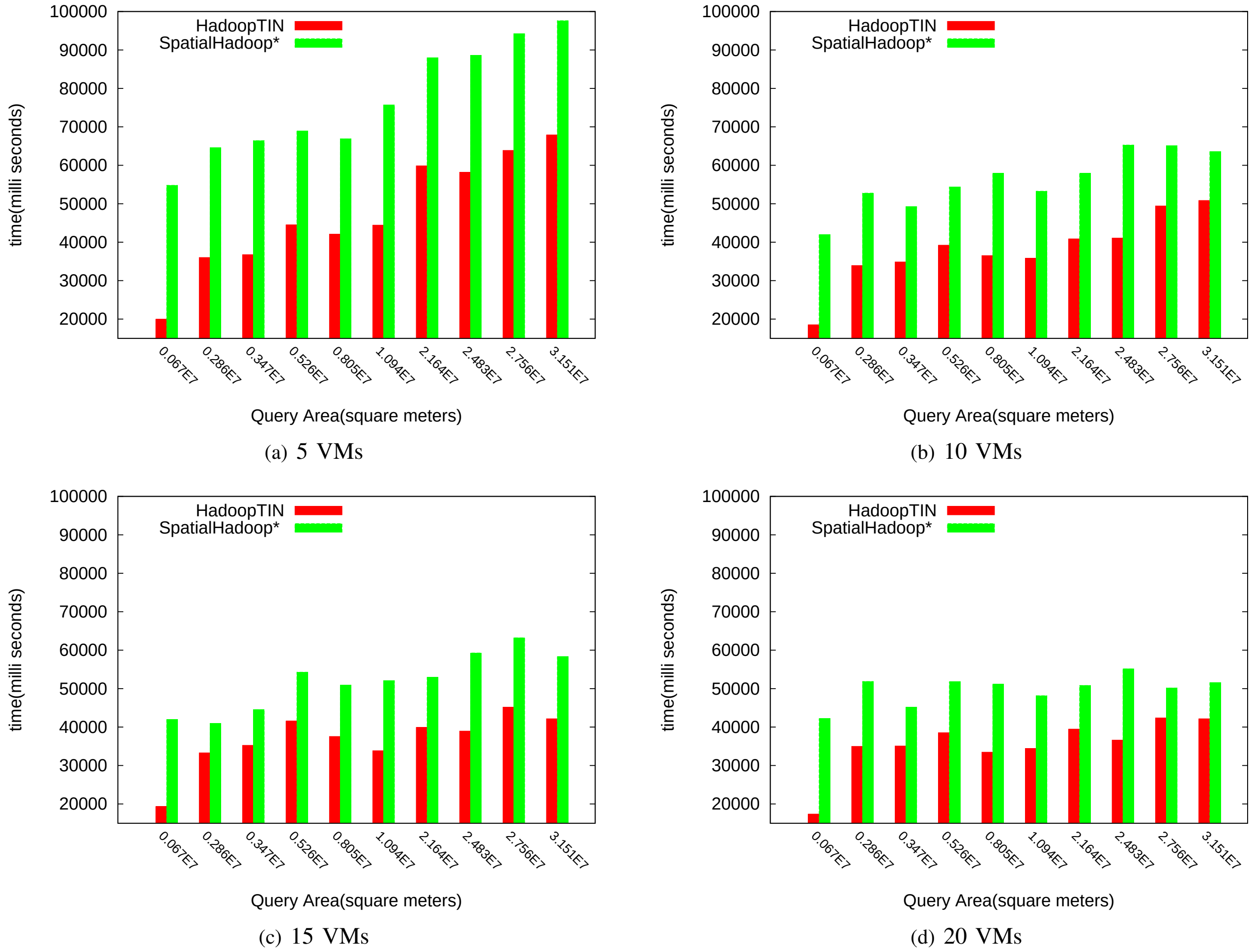
D. Quad-tree Based File Organization

Finally, we discuss our proposed quad-tree prefix based TIN file filtering strategy which tries to read in only the necessary TIN data rather than scanning the whole dataset stored in HDFS. Similar to how quad-tree organized by the prefix tree is used to minimize CPU load of the map tasks, we use a similar idea to reduce the amount of data processed by the polygon retrieval query. The core idea behind the proposed



QuadIndex	PointID	Latitude Longitude	Elevator	Adj_TriangleIDs
00000	8349	40.44207 -79.95496	156	324,12,45
00000	23	40.44778 -79.97353	154	7894,72
00120	45624	40.44594 -79.96281	240	6014,914,326
....				

Fig. 6: File Organization



(a) 5 VMs

(b) 10 VMs

(c) 15 VMs

(d) 20 VMs

Fig. 8: Execution Time for various query area size

approach is to separate the TIN data into fairly smaller files such that each file shares the same prefix. For instance, a large terrain's TIN data can be organized as multiple smaller files such as File_00, File_01, File_02 and so on, see Figure 3.4 for an example of a file with name File_00. After we organize the terrains file in this manner, we use it in the file filtering stage which scans only the required records to filter those files that are outside the query area. For example, in Figure 3(b), if we subdivide the TIN data into files based on a depth level of 2, we can see that the set of grid indices that intersect the query area correspond to $\{3, 23, 21, 12, 13, 03\ \}$. Hence, we only need to scan the files $\{30, 31, 32, 33, 23, 21, 12, 13, 03\}$ in the HDFS and thus minimizes the amount of data processed.

In practice we note that the length of the prefix constituting the file's name is an important parameter and it can affect the efficiency of the job to a certain extent. Specifically, the longer the prefix length becomes, the smaller is the size of the divided file so that more data can be filtered. However, we notice that Hadoop is not good at dealing with small sized files, especially when the files are less than the input split size. In order to balance this, we ensure that each TIN file size is a multiple of or equal to the Hadoop data block size. We compute the

Algorithm 3 Optimized MapReduce Polygon Retrieval

```
$quad\_index$: the quad index
$tin\_point$: a TIN point in space
$ptree$: the prefix tree
procedure MAP($quad\_index, tin\_point$)
    read prefix tree, $ptree$ from the global shared memory
    read the query area from the global shared memory
    search the prefix of $quad\_index$ in the prefix tree,
  $ptree$ until it ends at a leaf node
    if search returns null  then
        // it means that the grid is outside the query area
        return
    end if
    if leaf node is marked as "within" then
        output $tin\_point$
    else
        perform point within boundary computation
    end if
end procedure
```

prefix length l such that $\frac{filesize}{4^l} \approx 64$ MB, where $filesize$ is the size of the entire TIN dataset and 64 MB here corresponds to the default input split size in Hadoop.

V. Experimental Evaluation

In this section, we present the experimental evaluation of our distributed polygon retrieval algorithm. We divide this section into two parts. First, we introduce the dataset and the computing environment used in our experimental study. We then evaluate and compare with existing solutions how the time cost of the spatial query jobs grows as the randomly generated query areas get larger. Finally, we evaluate our algorithms on various sizes of Hadoop cluster to measure the efficacy of our approach across various cluster sizes.

A. Datasets and Experiment Environment

For our datasets, we use the LIDAR data of Pittsburgh city and convert it into TIN format with the help of the LASTool[22]. The data of Pittsburgh is originally divided into $5 * 5$ equally sized grid cells and each grid cell represents a terrain of 10000 metes * 10000 meters. There are 3 million points and 6 million triangles in each grid cell and the size of each grid's TIN file is approximately 500 MB. We conducted our experiments on a cluster of virtual machines created by OpenStack[8] hosted on a 5-node experimental cluster. Each server in the cluster has an Intel Xeon 2.2GHz 4 Core with 16 GB RAM and 1 TB hard drive at 7200 rpm. Each virtual machine in our setup had 1 VCPU with 2 GB RAM and 20 GB hard drive with Ubuntu Server 12.04(32 bit).

B. Time Cost vs Query Area

We first demonstrate how the time cost of the polygon retrieval algorithm grows as a function of the query area size. In this experiment, we use all of 4 grids' of data of Pittsburgh as input (2000 MB). For each polygon retrieval, we generated a polygon area for the query randomly. For comparing our results with existing distributed polygon retrieval techniques, we chose Spatial-Hadoop[29] as the benchmark. Since Spatial-Hadoop neither provides polygon retrieval nor supports the TIN data format directly, for our experiments, we modified their interfaces and we executed the polygon retrieval operation as suggested in the Spatial-Hadoop tutorial[29]. As suggested, we performed the point within polygon computation directly in the map function.

Figure 8(a) - Figure 8(d) show the relationship between the time cost and the polygon query area using a 5, 10, 15 and 20 VM cluster respectively. From the figures, we notice that as the query area gets larger, the time cost generally increases as more TIN data gets processed in the map and reduce phases. From the result, we also infer that our algorithm on an average runs 20%-50% faster than the existing technique in SpatialHadoop. As explained in Section IV, our proposed algorithm significantly avoids the geometry floating point computation in the map phase, especially when the query area is not very large and therefore, when the query area becomes larger, we notice that the I/O time dominates the CPU time and hence the CPU time savings become less significant.

C. Time Cost Vs Cluster size

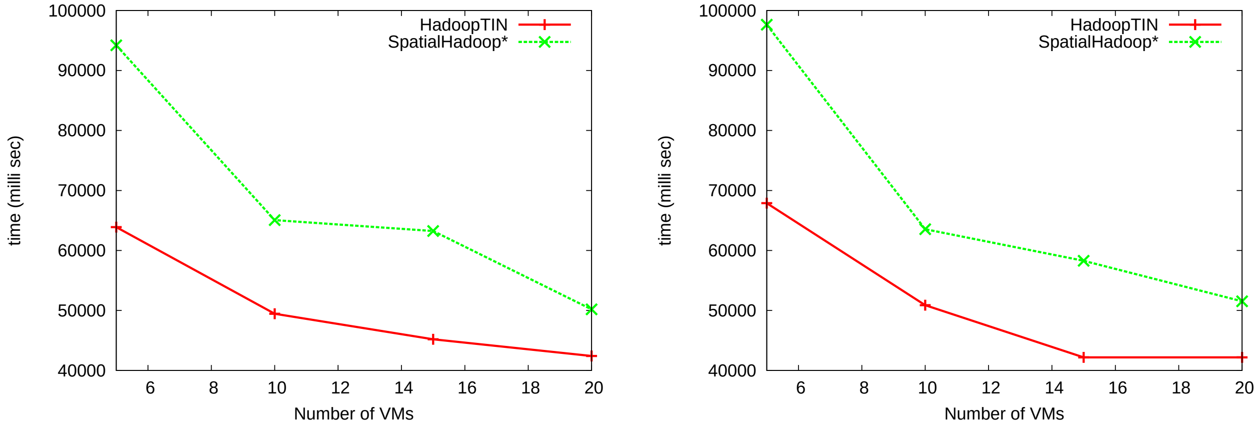
We next evaluate the effectiveness of our polygon retrieval algorithm by varying the size of the Hadoop cluster in terms of the number of VMs. For this experiment, we generated several random query shapes and used them to run queries on different cluster sizes. Figure 9(a) and Figure 9(b) show the time cost on various cluster sizes when the query area is $2.7e + 7m^2$ and $3.5e + 7m^2$ respectively. We infer from Figure 9(a) and Figure 9(b) that the execution time decreases gradually as the cluster size becomes larger. Overall, we find that the proposed technique scales well with the number of nodes in the Hadoop cluster showing a significant reduction in job execution time with increase in cluster size.

VI. Conclusion

In this paper, we presented a distributed polygon retrieval algorithm based on MapReduce to provide fast real-time responses to spatial queries over large-scale spatial datasets. Our proposed algorithm first hierarchically indexes the spatial terrain data using a quad-tree which filters out a significant amount of data in the pre-processing stage based on the query object. It then dynamically builds a prefix tree based on the quad-tree index to query the relationship between the terrain data and query area in real time which leads to significant savings in both I/O load and CPU time. The evaluation results of the techniques in a Hadoop cluster show that our techniques achieve significant reduction in job execution time for the queries and and shows a good scalability. As part of future work, we plan to develop distributed algorithms for other commonly used geospatial operations such as terrain visibility computation, flood simulation and 3D navigation.

References

[1] B. Igou "User Survey Analysis: Cloud-Computing Budgets Are Growing and Shifting; Traditional IT Services Providers Must Prepare or Perish". *Gartner Report*, 2010



(a) Execution time for an area of 2.7e+7

(b) Execution time for an area of 3.5e+7

Fig. 9: Impact of number of VMs

[2] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[3] Hadoop. http://hadoop.apache.org.

[4] Willard, D.E. "Polygon retrieval". *SIAM Journal on Computing 11.1 (1982): 149-165.*

[5] Mark de Berg, O.C., Marc van Kreveld, Mark Overmars. "Simplex Range Searching". *Computational Geometry*,2008

[6] Peucker, Thomas K., et al. "The triangulated irregular network". *Amer. Soc. Photogrammetry Proc. Digital Terrain Models Symposium. Vol. 516. 1978..*

[7] White, Tom. "Hadoop: The definitive guide". *O'Reilly Media, Inc.,2012.*

[8] Openstack: *http://www.openstack.org/*

[9] Google App Engine. "Basic Geospatial Queries in Google App Engine". https://code.google.com/p/geomodel/,2009

[10] Eldawy, Ahmed, and Mohamed F. Mokbel. "A demonstration of SpatialHadoop: an efficient mapreduce framework for spatial data". *Proceedings of the VLDB Endowment 6.12 (2013): 1230-1233.*

[11] Cormen, T., Stein, C., Rivest, R., and Leiserson, C. Introduction to Algorithms. In *McGraw-Hill Higher Education*.

[12] Sioutas, Spyros, et al. "Canonical polygon queries on the plane: A new approach". *arXiv preprint arXiv:0805.2681 (2008)*

[13] Samet, Hanan. The Quadtree and Related Hierarchical Data Structures. In *ACM Computing Surveys*, 1984

[14] Tung, Lun Hsing, and Irwin King. "A two-stage framework for polygon retrieval". Multimedia Tools and Applications 11.2 (2000): 235-255.

[15] Paterson, Michael S., and F. Frances Yao. "Point retrieval for polygons". *Journal of Algorithms 7.3 (1986): 441-447.*

[16] Karimi, Hassan Ali, Duangduen Roongpiboonsopit, and Haopeng Wang. "Exploring RealTime Geoprocessing in Cloud Computing: Navigation Services Case Study". *Transactions in GIS 15.5 (2011): 613-633.*

[17] Puri, Satish, et al. "MapReduce algorithms for GIS polygonal overlay processing" *Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), 2013 IEEE 27th International. IEEE, 2013..*

[18] Ji, Changqing, et al. "Inverted grid-based knn query processing with mapreduce". *ChinaGrid Annual Conference (ChinaGrid), 2012 Seventh. IEEE, 2012.*

[19] Akdogan, Afsin, et al. "Voronoi-based geospatial query processing with mapreduce". *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. IEEE, 2010.*

[20] Aji, Ablimit, et al. "Hadoop GIS: a high performance spatial data warehousing system over mapreduce". *Proceedings of the VLDB Endowment 6.11 (2013): 1009-1020.*

[21] Eldawy, Ahmed, et al. "CG Hadoop: computational geometry in MapReduce". *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2013.*

[22] Hanjianga, Xiong, Tang Limina, and Sun Longa. "A Strategy to Build a Seamless Multi-Scale TIN-DEM Database". *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 37.*

[23] Al-Salami, A. "TIN support in an open source spatial database". *MS Thesis, International Institute for Geoinformation Science and Earth Observation (ITC), Enschede, The Netherlands (2009).*

[24] Kothuri, Ravi, Albert Godfrind, and Euro Beinat. "Pro oracle spatial for oracle database 11g". *Berkeley: Apress, 2007.*

[25] Clementini, Eliseo, Jayant Sharma, and Max J. Egenhofer. "Modelling topological spatial relations: Strategies for query processing" . *Computers & graphics 18.6 (1994): 815-822.*

[26] Shimrat, Moshe. "Algorithm 112: position of point relative to polygon". *Communications of the ACM 5.8 (1962): 434.*

[27] Wikipedia. Trie. *//http://en.wikipedia.org/wiki/Trie*

[28] LAStools. *http://www.cs.unc.edu/ isenburg/lastools/*

[29] SpatialHadoop: *http://spatialhadoop.cs.umn.edu/operations.html*

[30] Lastools: *http://www.cs.unc.edu/ĩsenburg/lastools/.*