ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20170418094421
Original scientific paper

A MapReduce-Based Big Spatial Data Framework for Solving the Problem of Covering a Polygon with Orthogonal Rectangles

Süleyman EKEN, Ahmet SAYAR

Abstract: The polygon covering problem is an important class of problems in the area of computational geometry. There are slightly different versions of this problem depending on the types of polygons to be addressed. In this paper, we focus on finding an answer to a question of whether an orthogonal rectangle, or spatial query window, is fully covered by a set of orthogonal rectangles which are in smaller sizes. This problem is encountered in many application domains including object recognition/extraction/trace, spatial analyses, topological analyses, and augmented reality applications. In many real-world applications, in the cases of using traditional central computation techniques, working with real world data results in a performance bottlenecks. The work presented in this paper proposes a high performance MapReduce-based big data framework to solve the polygon covering problem in the cases of using a spatial query window and data are represented as a set of orthogonal rectangles. Orthogonal rectangular polygons are represented in the form of minimum bounding boxes. The spatial query windows are also called as range queries. The proposed spatial big data framework is evaluated in terms of horizontal scalability. In addition, efficiency and speed-up performance metrics for the proposed two algorithms are measured.

Keywords: big spatial data; GIS; MapReduce; polygon covering; spatial query

1 INTRODUCTION

In the polygon covering problem, the answer to the question of whether a polygon (or spatial query window) is fully covered by given polygon set is searched. In recent years, polygon covering process has been used in different areas including information systems such as Geographic Information Systems (GIS), Remote Sensing (RS), high performance computing systems, network monitoring, computational geometry, and large data management systems [1]. It is faced mostly in applications such as military sensor coverage and targeting [2, 3], spatial queries, telecommunications and wire and component layout in very-large-scale integration (VLSI), map overlay in GIS, motion planning, and collision detection. There are a number of polygon covering problems whose names change depending on the definition of polygon. In some cases, polygons become orthogonal polygons, also known as rectilinear. In some cases, polygons are covered with rectangles, polygons or even triangles. An orthogonal polygon is one whose edges are all aligned with a pair of orthogonal coordinates [4]. The work presented in this paper focuses on the coverage problems in orthogonal polygons.

There are two general types of data formats in GIS, these are raster and vector data types. While raster data (bitmap images) stores a grid of individual pixels in an image format, vectors utilize mathematical calculations between points and connection paths such as points, lines, and polygons [5]. In this paper, we firstly focus on orthogonal polygon covering problem and find a solution over vector based GIS data. Later on, we show how our solution to orthogonal polygon covering problem in vector domain can also be applied to satellite images in the raster format.

In many real-world applications, in the cases of using traditional central computation techniques, working with real world data results in a performance bottlenecks. When the data gets so large in volume it can be challenging to actually make use of the information. The orthogonal polygon covering analyses on large volumes of spatial data takes many hours by using standard centralized computing nodes. In this study, we propose Hadoop MapReduce based distributed polygon covering algorithms on a set of distributed heterogeneous computing nodes [6]. Hadoop framework and its MapReduce computation paradigm supports many key features such as processing data in scalable manner, fault-tolerance to crashes, omissions, and arbitrary failures. With the success of MapReduce, a number of spatial query systems [7-9] and frameworks [10, 11] have emerged to enable large scale spatial query processing on large datasets in the literature. The following section gives detailed explanations on the relevant works.

Not all the application utilizes Hadoop and MapReduce computation paradigm. Application developers need to remodel and redefine the data and the related computation processes according to the distributed computation and MapReduce paradigms. In this paper we propose a MapReduce-based big spatial data framework for solving the problem of covering an orthogonal polygon with the set of orthogonal rectangles. To do so, we remodel the data sets and redefine the related computation processes in accordance with the MapReduce paradigm.

The rest of this paper is organized as follows. Section 2 reviews research on using Hadoop to process big spatial data. Section 3 presents problem definition. Section 4 describes MapReduce based algorithms. Section 5 explains experimental setup, test scenarios, and performance results. Section 6 concludes the paper and describes the directions for future work.

2 RELATED WORKS

Hadoop, in its pure version, does not support features suited for spatial data and its analyses. Therefore, some researchers have focused on solving this issue by extending Hadoop API with the required functionalities for spatial data analysis. In addition, implementation of spatial queries using these extended distributed technologies is particularly a challenging task. In general, researchers have addressed different kinds of queries such as range queries [12], kNN and reverse nearest neighbour (RNN) query

[13], different kinds of spatial join operations such as binary [14] and multi-way join [15], skyline [16], and convex hull [17]. According to the query types, query pipelines might be differentiated in MapReduce. Spatial queries can be realized with different Hadoop's MapReduce patterns depending on query complexity. A research group in University of Minnesota has developed SpatialHadoop [18], which is an extension to the core Hadoop framework. Hadoop-GIS [19], which is another big data framework, is a spatial query system over MapReduce. Hadoop-GIS has capability of processing spatial queries with spatial query engine. In addition, it enables data and space based partitioning and pipelined queries. Wang et al. [20] have proposed query engine called Resque to support high performance spatial queries and analytics for compute intensive spatial data on MapReduce and CPU-GPU based hybrid platforms. Migliorini et al. [21] have extended the GeoUML methodology to produce Pigeon, which is a data-flow language defined on top of Spatial Hadoop for validation procedures. So, validation of large datasets can be made in a feasible way. The works mentioned so far are all related to defining an overlay API on top of Hadoop framework for enabling spatial data analyses. There are some other researches on performance optimization of spatial data analyses.

There are also some other works specifically on orthogonal rectangles based problems such as polygon intersection/overlay. Eken and Sayar [22] have proposed two solutions to handle polygon intersection (mosaic selection problem). They have overcome mosaic selection problem by means of finding rectangular sub-regions intersecting with range query. Mosaic images are represented with MBRs. The former approach is based on hybrid of Apache Hadoop and HBase and the latter one is based on Apache Lucene. In both approaches, vertical scalability (different data sizes) is considered instead of horizontal scalability. In extended version of [22], one MapReduce based algorithm is proposed for finding all objects that overlap a given range query [23]. They show that running times for orthogonal polygon overlay process reduce with increase of conventional machines. However, to the best of our knowledge, there is no work on tackling scalability problem on polygon covering with orthogonal rectangles. The aim of this study is introducing a novel high performance MapReduce-based big data framework to solve the polygon covering problem in the cases of using a spatial query window and data are represented as a set of orthogonal rectangles. Feasibility and efficiency of the proposed framework are proven to be working first on synthetic data, and then, on real-world satellite mosaic image data. Proposed algorithms are promising on big spatial data; the latter one especially gives better results for millions of polygons.

3 PROBLEM DEFINITION

The aim is to determine if a given orthogonal rectangle, or a query window, is covered by a given set of orthogonal rectangles. The problem is illustrated in Fig. 1. Throughout the paper, T represents a query window, p_i, $i > 0$, represents individual orthogonal rectangles in the set (see Fig. 1).

Definition 3.1. (Minimum Bounding Rectangle - MBR) An expression of the maximum extents of a 2-dimensional object or set of objects within its (or their) 2D (x, y) coordinate system, in other words $\min_x$, $\max_x$, $\min_y$, $\max_y$.

Definition 3.2. (Set operations between two rectangles) Let A and B be two rectangles in 2D space. Set operations on A and B can be defined as follows:

$$A \cup B = \{(x, y) \mid (x, y) \in A \text{ or } (x, y) \in B\}$$
$$A \cap B = \{(x, y) \mid (x, y) \in A \text{ and } (x, y) \in B\}$$
$$A - B = \text{clos}(\{(x, y) \mid (x, y) \in A \text{ and } (x, y) \notin B\})$$

Definition 3.3. (Polygon Covering with Rectangles) Let's consider m number of polygons in a given polygon set P. $P = \{p_1, p_2, ..., p_m\}$ and target polygon is T. T is user defined query region and also called as query window. The orthogonal rectangle covering problem seeks if the below condition is met.

$$T \subseteq P_1 \cup ... \cup P_m \quad (1)$$

Fig. 1 illustrates the covering problem. Fig. 1 is an example for $m = 3$. Fig. 1(a) shows that target region (map) is fully covered by rectangles p_1, p_2, and p_3. However, target region is not fully covered in Fig. 1(b). The yellow regions indicate uncovered parts of T.

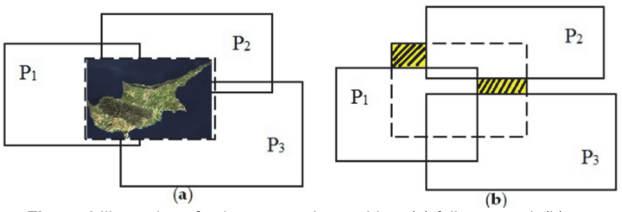

Figure 1 Illustration of polygon covering problem (a) fully covered, (b) not covered

Pseudo-code for the sequential polygon covering algorithm is as following:

Algorithm: Sequential polygon covering algorithm
Input: target rectangle (t) coordinates, polygon set
Output: true/false
1. begin
2. create a matrix of [t.maxx+1][t.maxy+1] size with all values 0
3. for each polygon *p* in polygon set do
4. if(!(t.mix>p.maxx)) && !(p.minx>t.maxx) && !(t.miny>p.maxy && !(p.miny>t.maxy)) then replace values of matrix at intersected parts to 1
5. end for
6. if all elements of matrix equal 1 then true
7. else false
8. end

Table 1 Notations and their meanings

Notations	Meaning
$(\min_x, \min_y)$	Lower left coordinates of MBR
$(\max_x, \max_y)$	Upper right coordinates of MBR
A and B	Any two rectangles
$P = \{p_1, p_2, ..., p_m\}$	Set of orthogonal rectangles
T	Target polygon or query window

Tab. 1 summarizes the notations used in this paper.

4 MAPREDUCE BASED POLYGON COVERING FRAMEWORK

Apache Hadoop which is a distributed open source project consists of two main parts: (i) Hadoop Distributed File System (HDFS) which stores input and output data files and (ii) Hadoop MapReduce parallel programming paradigm which is a promising solution to large-scale data applications. A Job in MapReduce is run in parallel and contains three phases: map, shuffle, and reduce. Data are represented as (key, value) pairs for MapReduce to process them. In map phase, map function produces intermediate key-value pairs. In reduce phase, the reduce function generates the final output pairs from intermediate key-value pairs produced by map phases [24].

The general architecture for polygon covering is illustrated in Fig. 2. The user can do polygon covering analysis on MBRs stored HDFS.

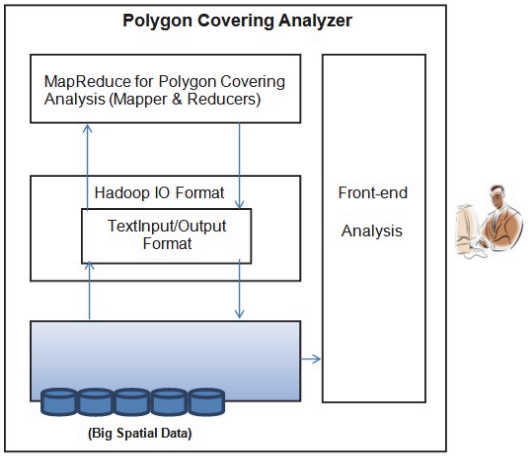

Figure 2 Distributed polygon covering framework

Here, we implement Hadoop MapReduce-based two different covering algorithms. Their detailed explanations will be given in the following paragraphs. Fig. 3 shows the sub-steps (input, splitting, mapping, reducing, and output) of one of our proposed algorithm (here, Algorithm 1 is exemplified).

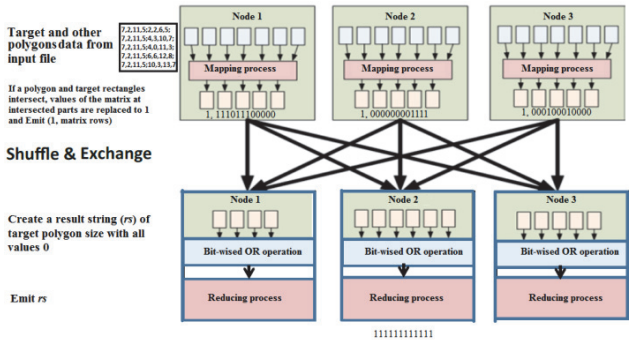

Figure 3 High-level representation of proposed Algorithm 1

Pseudo-code for the Algorithm 1 is as the following:

Algorithm 1: MapReduce polygon covering algorithm

method MAP

Input: key: input line id; **value:** line content (rectangle coordinates)
Output:key: 1; **value:** strings (s) created from sub-matrix rows of intersected polygon coordinates
1. **begin**
2. split a line and extracts coordinates of a polygon as $p.min_x$, $p.min_y$, $p.max_x$, and $p.max_y$
3. create a matrix of $[t.max_x+1][\ t.max_y+1]$ size with all values 0
4. **if**($!(t.min_x>p.max_x)$) && $!(p.min_x>t.max_x)$ && $!(t.min_y>p.max_y$&& $!(p.min_y>t.max_y)$) **then** replace values of matrix at intersected parts to 1
5. arrange matrix rows side by side as a string (s) and send this string to reduce phase
6. **output** (1, s)
7. **end**

Method REDUCE
Input: key: 1; **value:** various string parts [s_1, s_2, . .] from mappers
Output:key: null; **value:** 1 or 0 according to bit-wised OR operation
1. **begin**
2. create a result string (*rs*) of $[t.max_x+1]\times[\ t.max_y+1]$ size with all values 0
3. **for each** (string s from[s_1, s_2, . .]) **do**
4. *rs* is bit-wised OR with s
5. **end for**
6. **output** (*rs*)
7. **end**

Algorithm 1 takes set of polygons in an input file. The polygons are orthogonal rectangles and represented with their MBRs ($p.min_x$, $p.min_y$, $p.max_x$, and $p.max_y$). The coordinates of target polygon, also query window, is ($t.min_x$, $t.min_y$, $t.max_x$, and $t.max_y$) and defined by a user. Every mapper receives 64 MB file split as input and emits it as key-value pairs.

In mapper phase, a zero matrix of size $[t.max_x+1][t.max_y+1]$ is created. If a polygon and the target rectangle intersect, values of the matrix at intersected parts are replaced to 1 and then the matrix rows are arranged side by side as a string and this string is sent to the reduce phase. In the reduce phase, a result string of length $[t.max_x\times t.max_y]$ is created with all values set to 0. This string is bit-wised OR with output of every mapper. Finally, values of result string are bit-wised AND. If the result equals 1, user defined query region/target polygon is fully covered by the other polygons, otherwise vice versa.

The disadvantage of Algorithm 1 is that it uses a matrix in a map phase and a string in a reduce phase. The algorithm does not perform well for big sized target polygons. To overcome this problem, we propose Algorithm 2 as the following:

Algorithm 2: MapReduce polygon covering algorithm

method MAP

Input: key: input line id; **value:** line content (polygon coordinates)
Output:key: 1; **value:** coordinates of intersected rectangle (($coor_{start\text{-}x}$, $coor_{start\text{-}y}$), ($coor_{finish\text{-}x}$, $coor_{finish\text{-}y}$))
1. **begin**
2. obtain rectangle coordinates from input

3. **if** an orthogonal polygon and a target rectangle intersect **then** send the coordinates of intersected region to reduce phase
4. **end if**
5. **output**(1, (($coor_{start-x}$, $coor_{start-y}$),($coor_{finish-x}$, $coor_{finish-y}$)))
6. **end**

Method REDUCE
Input: key: 1; **value:** list of coordinates c in form of (($coor_{start-x}$, $coor_{start-y}$),($coor_{finish-x}$, $coor_{finish-y}$))
Output:key: null; **value:** 1 or 0 according to bit-wised AND operation
1. begin
2. create a result string (*rs*) of [t.maxx+1]×[t.maxy+1] size with all values 0
3. **for each** (coordinates c from[[c_1, c_2], . .]) **do**
4. replace values of *rs* with 1 according to starting and finishing coordinates of
intersected region
5. end for
6. **output** (*rs*)
7. end

The input file to Algorithm 2 is set of rectangular polygons with their MBRs as in Algorithm 1. The coordinates of the target polygon are again specified by a user. Every mapper receives 64 MB file split as input and emits it as key-value pairs. In the mapper phase, if a polygon and the target rectangle intersect, coordinates of intersected polygons are sent to the reduce phase. In the reduce phase, a result string of length [$t.max_x \times t.max_y$] size is created with all values 0. Values of this string which are intersected parts are replaced with 1 according to output of every mapper. At the end of reduce phase, values of result string are bit-wised AND. If the result equals 1, user defined query region/target polygon is fully covered by the other polygons, otherwise vice versa.

Time complexities of the proposed algorithms are as the following: The map step of Algorithm 1 sets the values of subparts of orthogonal rectangles which intersects with target rectangle to 1. Let the dimensions of matrix be $N \times N$. So, time complexity of this process would be $O(N^2)$. Also, arranging matrix rows side by side as a string would be $O(N^2)$. The reduce step of Algorithm 1 does bit-wised OR with string with length $N \times N$. The overall time complexity of algorithm 1 is $O(3N^2)$. When N is large, constant factor can be ignored to get to the simplest expression. So, $O(3N^2)$ becomes just $O(N^2)$. The map step of Algorithm 2 tests whether an orthogonal polygon and a target rectangle intersect and then sends the coordinates of intersected region to the reduce phase. This step takes constant time. The reduce step of Algorithm 2 replaces values of *result string* (*rs*) with 1 according to starting and finishing coordinates of intersected region. The overall time complexity of Algorithm 1 is $O(c + N^2)$, and it becomes $O(N^2)$. This proves that Algorithm 2 is faster than Algorithm 1.

5 EXPERIMENTAL RESULTS AND EVALUATION

5.1 Experimental Setup

The research experiment is set up to evaluate the algorithms' efficiencies. The experiment is set up on a cluster of nine nodes i.e., one name node/Job tracker (master) and eight data nodes/task tracker (slaves). The framework developed for distributed polygon covering processing with Hadoop was tested by creating Hadoop clusters on the OpenStack [25] installed cloud system. The characteristics of the virtual computers created with the OpenStack system are given in Tab. 2. The master computer has 8 VCPU, 160 GB disk and 8 GB RAM. Each worker computer has 4 VCPUs with 80 GB disk space and 8 GB RAM resources. We control cluster with Hortonworks Data Platform (HDP) on OpenStack using Apache Ambari.

Table 2 Characteristics of virtual nodes

Node name	VCPU	Disk	RAM
test-master-large-0	8	80 GB	8 GB
test-worker-medium-0	4	40 GB	4 GB
test-worker-medium-1	4	40 GB	4 GB
test-worker-medium-2	4	40 GB	4 GB
test-worker-medium-3	4	40 GB	4 GB
test-worker-medium-4	4	40 GB	4 GB
test-worker-medium-5	4	40 GB	4 GB
test-worker-medium-6	4	40 GB	4 GB
test-worker-medium-7	4	40 GB	4 GB

5.2 Validation of Algorithms with Synthetic Data

All polygons' MBR coordinates are created randomly as an input file. They are defined in 2D space. Input files are stored on HDFS for further processing. The number of polygons varies from 1000 to 10 million and size of all polygons is 7000×7000. After creating orthogonal rectangle dataset randomly, proposed polygon covering algorithms are evaluated on the dataset for relatively small and relatively large target rectangle. Test scenarios are listed as below. All scenarios are experimented using nine commodity machines.

(i) Case 1: Small target rectangle (500×500) over data space using Algorithm 1
(ii) Case 2: Small target rectangle (500×500) over data space using Algorithm 2
(iii) Case 3: Large target rectangle (7000×7000) over data space using Algorithm 1
(iv) Case 4: Large target rectangle (7000×7000) over data space using Algorithm 2

Figs. 4 and 6 show the horizontal scalability of Algorithm 1 for small and large target rectangle. Figs. 5 and 7 show the horizontal scalability of Algorithm 2 for small and large target rectangle. Algorithm 2 is faster than Algorithm 1. Because the coordinates of intersected region are sent to reduce phase in Algorithm 2 instead of sending all data. Also, we investigate scalability of window size under constant number of polygon number (10*K*) for Algorithm 2. Tab. 3 shows the linear increment according to the change of window size.

Table 3 Scalability analysis according to window size

Window size	Processing time (sec)
500	17
2500	24
5000	78
7000	110
10000	256

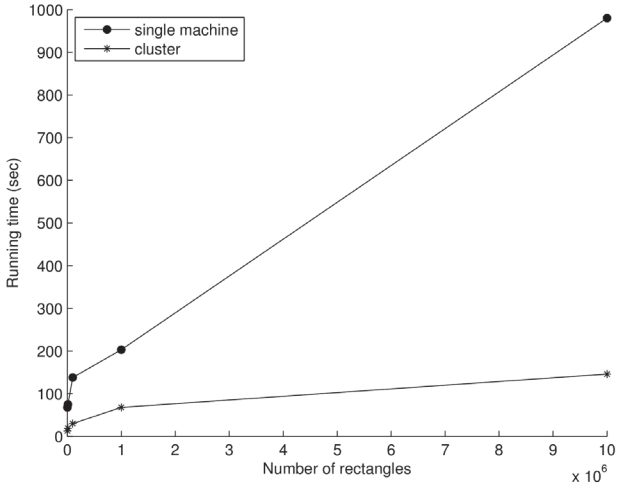

Figure 4 Horizontal scalability analysis of Algorithm 1 for small target rectangle

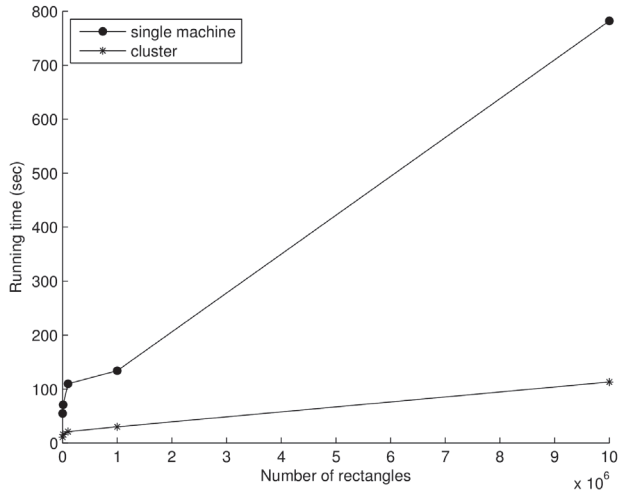

Figure 5 Horizontal scalability analysis of Algorithm 2 for small target rectangle

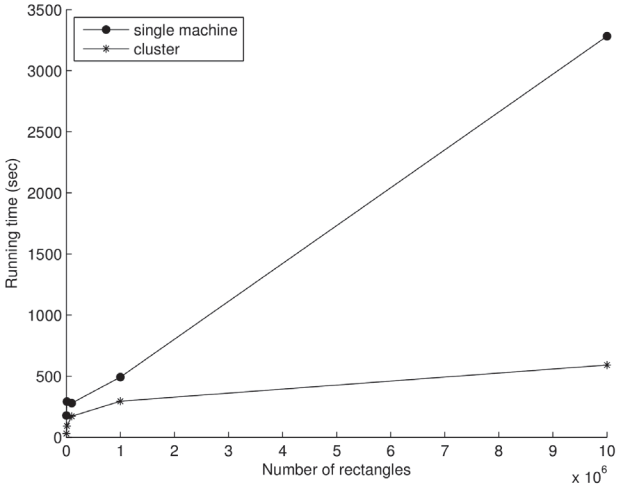

Figure 6 Horizontal scalability analysis of Algorithm 1 for large target rectangle

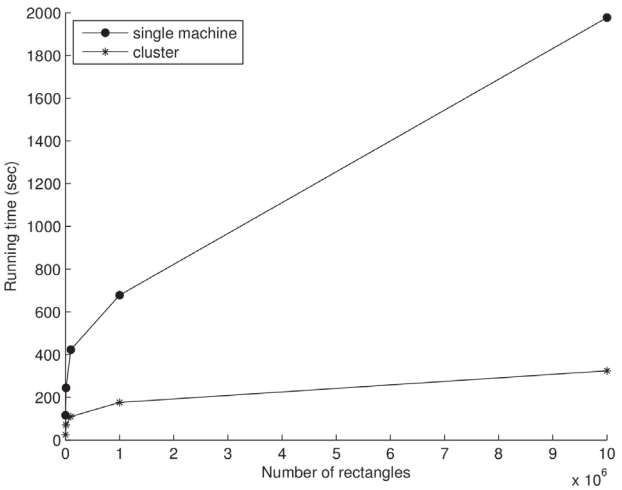

Figure 7 Horizontal scalability analysis of Algorithm 2 for large target rectangle

A program's processing time (T) indicates the time that a program takes to compute the answer to the question. This time depends on different parameters such as hardware characteristics, the algorithm, and program implementation style. Speed-up also is the effect of applying an increasing number of resources to a fixed amount of work to achieve a proportional reduction in execution times. Ideally, a parallel program should run twice as fast on two processors as on one processor, three times as fast on three processors, four times as fast on four processors, and so on. Thus, ideally, speedup should be equal to number of processors. When $10K$ rectangles are used, speed-up values of algorithm 1 using small and large target rectangle are 6.71 (980 sec / 146 sec) and 5.54 (3282 sec / 592 sec), respectively. Speed-up values of algorithm 2 using small and large target rectangle are 6.92 (782 sec / 113 sec) and 6.10 (1977 sec / 324 sec), respectively. While cluster has nine machines, speed-up is sub-linear. Because speed-up values are less than number of resources (here, $K = 9$).

Efficiency is a metric that captures how close to ideal is a program's speedup. It can be computed by dividing the speed-up value by the number of processors. An ideal parallel program has an efficiency of 1. So, algorithm 2 is closer to ideal than algorithm 1.

5.3 Analyzing Algorithms with Real-World Dataset

The feature extraction from satellite images, such as roads and bridges, is significant research area of remote sensing. Feature extraction from satellite images requires to merge images obtained at different times mostly captured by sensors. This process is called image registration or stitching. Especially, image to image registration is done to match two or more images taken, for example, at different times, or from different viewpoints.

To show workability of proposed algorithms for real-life applications, we choose registration problem of satellite images. Before image registration process, it is important to determine whether there are enough mosaic images for interested objects. Polygon covering problem with orthogonal rectangles comes in here. Datasets on our experiments are obtained by the recently launched LandSat-8 satellite. We store the tiles and their metadata (geographic corner coordinates as NW, NE, SW and SE) in

HBase (https://hbase.apache.org/) distributed database which can be easily processed by Hadoop. Target polygon is specified by a user via Google Maps. Proposed polygon covering algorithms were evaluated to check whether all tiles are enough to merge user defined region/target polygon or not. Table 4 shows the scalability analysis of the proposed algorithms on LandSat-8 satellite dataset.

Table 4 Scalability analysis on LandSat-8 satellite images

	Running time (msec)	
# of mosaics	Algorithm 1	Algorithm 2
4	124	102
10	315	299
20	480	364
40	862	710
80	2203	1965

6 CONCLUSION AND FURTHER WORKS

When a large scale of data is considered for polygon covering problem, it is time consuming process. Because it is CPU and memory intensive operation. After reviewing the literature on GIS polygon covering problem, we have not encountered such a work on Hadoop MapReduce-based parallel or distributed polygon covering computation on a cluster.

In this paper, we have analyzed the feasibility and efficiency of proposed polygon covering algorithms on a synthetic and real-life dataset in a distributed and parallel manner. Synthetic data is created using some random functions. Real-life data is captured from the LandSat-8 satellite. We have experimented the performance of these algorithms. Algorithm 2 outperforms Algorithm 1. We also test Algorithm 2 in terms of window size scalability. Experimental results show that proposed polygon covering algorithms discussed in this paper can be applicable for parallelizing similar GIS operators such as overlay, union, and intersection etc. In the future, we will study decreasing processing time for polygon covering problem.

Acknowledgements

This work has been supported by the TUBITAK under grant 215E189. The Hadoop infrastructure used in the study is provided by TUBITAK B3LAB.

7 REFERENCES

[1] Agrawal, D., Das, S., & El Abbadi, A. (2011). Big Data and Cloud Computing: Current State and Future Opportunities. In *Proceedings of 14th International Conference on Extending Database Technology*, 530-533. https://doi.org/10.1145/1951365.1951432

[2] Giachetta, R. (2013). AEGIS: A state-of-the-art spatio-temporal framework for education and research. *OSGeo Journal, 13*(1), 68-77.

[3] Daniels, K. & Inkulu, R. (2001). Translational polygon covering using intersection graphs. *Proceedings of 13th Canadian Conference on Computational Geometry*, 61-64.

[4] Puri, S. (2015). Efficient Parallel and Distributed Algorithms for GIS Polygon Overlay Processing. *PhD Thesis*, Georgia State University, Atlanta, Georgia, U.S.

[5] Eken, S. & Sayar, A. (2015). An Automated Technique to Determine Spatiotemporal Changes in Satellite Island Images with Vectorization and Spatial Queries. *Sadhana - Academy Proceedings in Engineering Science*, 40, 121-137.

[6] Dean, J. & Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Commun.,* 51, 107-113. https://doi.org/10.1145/1327452.1327492

[7] Aji, A., Wang, F., & Saltz, J. H. (2012). Towards Building A High Performance Spatial Query System for Large Scale Medical Imaging Data. *Proceedings of 20th International Conference on Advances in Geographic Information Systems*, 309-318. https://doi.org/10.1145/2424321.2424361

[8] Lu, J. & Guting, R. H. (2013). Parallel SECONDO: Practical and efficient mobility data processing in the cloud. *Proceedings of 2013 IEEE International Conference on Big Data*, 107-125. https://doi.org/10.1109/BigData.2013.6691767

[9] Ray, S., Simion, B., Brow, A. D., & Johnson, R. (2013). A parallel spatial data analysis infrastructure for the cloud. *Proceedings of 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 274-283. https://doi.org/10.1145/2525314.2525347

[10] Aji, A., et al. (2013). Demonstration of Hadoop-GIS: A Spatial Data Warehousing System over MapReduce. *Proceedings of 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 518-521. https://doi.org/10.1145/2525314.2525320

[11] Eldawy, A., Li, Y., Mokbel, M. F., & Janardan, R. (2013). CG-Hadoop: Computational geometry in MapReduce. *Proceedings of 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS 2013*, 284-293. https://doi.org/10.1145/2525314.2525349

[12] Whitman, R. T., Park, M. B., Ambrose, S. A., & Hoel, E. G. (2014). Spatial Indexing and Analytics on Hadoop. *Proceedings of 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 73-82. https://doi.org/10.1145/2666310.2666387

[13] Akdogan, A., Demiryurek, U., Banaei-Kashani, F., & Shahabi, C. (2011). Voronoi-Based Geospatial Query Processing with MapReduce. *Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010*, 9-16. https://doi.org/10.1109/CloudCom.2010.92

[14] Zhang, S., Han, J., Liu, Z., Wang, K., & Xu, Z. (2009). SJMR: Parallelizing spatial join with MapReduce on clusters. *2009 IEEE International Conference on Cluster Computing and Workshops*, New Orleans, LA, 1-8. https://doi.org/10.1109/CLUSTR.2009.5289178

[15] Gupta, H. et al. (2013). Processing Multi-way Spatial Joins on Map-reduce. *Proceedings of the 16th International Conference on Extending Database Technology, EDBT '13*, 113-124. https://doi.org/10.1145/2452376.2452390

[16] Eldawy, A., Li, Y., Mokbel, M. F. and Janardan, R. (2013). CG_Hadoop: Computational Geometry in MapReduce. In *SIGSPATIAL*. https://doi.org/10.1145/2525314.2525349

[17] Zhang, S., Han, J., Liu, Z., Wang, K., & Feng, S. (2009). Spatial Queries Evaluation with MapReduce. *Proceedings of Eighth International Conference on Grid and Cooperative Computing*, 287-292. https://doi.org/10.1109/GCC.2009.16

[18] Eldawy, A. & Mokbel, M. F. (2013). A demonstration of SpatialHadoop: an efficient MapReduc framework for spatial data. *VLDB Endow.,* 6, 1230-1233. https://doi.org/10.14778/2536274.2536283

[19] Aji, A., et al. (2013). Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *VLDB Endow.,* 6, 1009-1020. https://doi.org/10.14778/2536222.2536227

[20] Wang, F., Aji, A., & Vo, H. (2014). High performance spatial queries for spatial big data: from medical imaging to GIS. *SIGSPATIAL Special,* 6, 11-18. https://doi.org/10.1145/2766196.2766199

[21] Migliorini, S., Belussi, A., Negri, M., & Pelagatti, M. (2016). Towards massive spatial data validation with SpatialHadoop. *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, 18-27. https://doi.org/10.1145/3006386.3006392
[22] Eken, S., Sayar, A. (2015). Big data frameworks for efficient range queries to extract interested rectangular sub regions. *International Journal of Computer Applications,* 119, 36–39. https://doi.org/10.5120/21372-4423
[23] Eken, S., Kızgındere, U., & Sayar, A. (2017). MapReduce Based Scalable Range Query Architecture for Big Spatial Data. In: Ivan I, Singleton A, Horák J, Inspektor T, editors. *The Rise of Big Spatial Data*. Switzerland: Springer International Publishing, 263-272. https://doi.org/10.1007/978-3-319-45123-7_19
[24] White, T. (2015). *Hadoop: The Definitive Guide*. 4th ed. Sebastopol, USA: O'Reilly Media.
[25] OpenStack. Available: http://www.openstack.org/

Contact information:

Süleyman EKEN, PhD Student
(Corresponding author)
Department of Computer Engineering
Kocaeli University
Umuttepe Campus, 41380, İzmit, TURKEY
suleyman.eken@kocaeli.edu.tr

Ahmet SAYAR, PhD
Department of Computer Engineering
Kocaeli University
Umuttepe Campus, 41380, İzmit, TURKEY
ahmet.sayar@kocaeli.edu.tr