A COMPARISON BETWEEN THE HADOOP AND SPARK DISTRIBUTED FRAMEWORKS IN THE CONTEXT OF REGION-GROWING SEGMENTATION OF REMOTE SENSING IMAGES

R. B. Andrade[1], J. M. F. Santos[1], G. A. O. P. Costa[1,*], G. L. A. Mota[1], P. N. Happ[2], R. Q. Feitosa[2]

[1] Institute of Mathematics and Statistics, Rio de Janeiro State University, Rio de Janeiro, Brazil
- (renanbides, josematheusuerj)@gmail.com, (gilson.costa, guimota)@ime.uerj.br
[2] Dept. of Electrical Engineering, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
- (patrick, raul)@ele.puc-rio.br

ICWG II/III: Pattern Analysis in Remote Sensing

KEY WORDS: Remote Sensing, Image Segmentation, Distributed Processing, Mapreduce, Hadoop, Spark

ABSTRACT:

This work follows a line of research dedicated to the parallelization of image segmentation algorithms on distributed computing environments, which is motivated by the increasing resolutions and availability of Remote Sensing (RS) images. Here we focus on region-growing segmentation, which is regarded as a time consuming and demanding approach in terms of computational resources. Its parallelization is a complex problem since it usually affects the final outcome in comparison to what would be delivered by a sequential solution. This is due to the fact that subdividing an image to perform segmentation of its tiles concurrently usually introduces undesirable artifacts near to the borders of the image tiles. Additional processing steps are then required to properly stitch together the segments alongside tiles borders in order to eliminate such artifacts. In this work we evaluated alternative implementations of a previously proposed region-growing distributed segmentation approach, which was originally built on top of the Hadoop distributed computing framework. We developed a new implementation of the approach, which was built with the Spark framework, and compared its performance with that of the original implementation. In this investigation RS images of various sizes were processed using different configurations of a physical computer cluster. We evaluated computational performances and accessed the differences among the segmentation outcomes generated by the alternative implementations. We also assessed the stability of the implementations by comparing the segmentations produced with different cluster configurations. Although the approach is, in principle, suitable to any region growing algorithm, the experiments were performed with a particular segmentation method, and the results showed that the Spark implementation consistently outperformed the Hadoop counterpart, bringing in most cases a significant improvement in terms of processing time. The experiment results also attested the stability of the distributed segmentation approach, as very similar results were produced with the alternative implementations, running on different cluster configurations.

1. INTRODUCTION

Considering the current rate of change of the Earth's surface, produced directly or induced by human activity, and the growing frequency of extreme environmental effects related to global warming; efficient methods for Remote Sensing (RS) data analysis are of utmost importance in a variety of application fields, such as environmental and urban monitoring, disaster response, food security, among others.

Advances in Earth observation technologies were responsible for increasing, at a very fast pace, the availability of data that can be used to study, predict and mitigate problems associated with these new environmental conditions. An increasing number of aerial and orbital systems are currently producing a great amount of input for those purposes. Illustrative examples are ESA's Sentinel Data Access service, which was, by the end of 2017, publishing around 10 TB of data daily (Castriotta , Knowelden, 2018), and NASA's EOSDIS Project, which currently adds about 6.4 TB of data to its archives and distributes almost 28 TB worth of data every day (Blumenfeld, 2019).

This scenario, however, leads to challenges related to the capacity of handling huge volumes of data, with respect to computational techniques and resources (Lee , Kang, 2017). There is, therefore, an important demand for automatic tools for interpreting RS images in a robust and scalable way.

Such increasing rate in digital data collection and the consequent demand for efficient data processing techniques capable of handling very large datasets is, however, not exclusive to Remote Sensing. Different information technology solutions have been devised to tackle this problem, and most of those initiatives are based on distributed processing: in which the datasets are divided into smaller sets that are processed independently, on different computing units.

Recently the authors of (Happ et al., 2016) proposed a novel approach for handling region-growing segmentation in a distributed way. Such approach enables distributed processing of very large RS images in a physical or virtual cluster, e.g., using cloud computing infrastructure. In this approach the image to be segmented is divided into tiles, which are indexed according to a particular indexing technique and processed independently on the various cluster nodes. After independent processing, a hierarchical stitching mechanism is employed in order to suppress segmentation artifacts along the borders of the tiles. Experiments conducted with an implementation based

*Corresponding author

on the MapReduce distributed programming model (Dean , Ghemawa, 2008) demonstrated the robustness and scalability of the approach. The solution was built over Apache Hadoop (Apache Hadoop Development Team, 2019), a widely used open-source implementation of the MapReduce model.

Meanwhile, alternatives to the MapReduce model have been proposed, in particular the Apache Spark distributed computing framework (Apache Spark Development Team, 2019), which has attracted much attention in the last few years, mostly because of its capacity to outperform Hadoop in many applications. And this is due to its ability of sharing memory among cluster nodes, instead of restricting inter-node communication to data file access, as it the case in Hadoop and in alternative distributed frameworks such as Apache Tez (Saha et al., 2015).

In this work we developed and evaluated a new implementation of the distributed segmentation approach proposed in (Happ et al., 2016) built on top of the Spark framework, and compared its performance with that of the original implementation. In this investigation we carried out a number of experiments in which RS images of various sizes were processed using different configurations of a physical computer cluster. We evaluated not only computational performance, but also accessed the differences among the segmentation outcomes generated by the alternative implementations. Furthermore, we assessed the stability of the implementations, by comparing the segmentations produced with different cluster configurations.

The remainder of this paper is organized as follows. In the next section, we indicate some related works. In Section 3 the distributed segmentation approach is briefly described, and in Section 4 we comment on the differences of the alternative distributed frameworks. In Section 5 we describe the experimental analysis, and in Section 6 we present conclusions and directions for future work.

2. RELATED WORK

Region-growing image segmentation is considered an expensive procedure in terms of processing time. This has to do with the fact that, at least in the most sophisticated and popular algorithms such as the ones proposed in (Baatz , Schäpe, 2000) and (Câmara et al., 1996), for two adjacent regions to be merged, all their neighbors have to be inspected. Also, in each of the various iterations of the algorithms, all regions must be visited.

Such computational burden has inspired a number of parallel image segmentation algorithms, ranging from traditional data-parallel approaches to GPU implementations. The work (Barder et al., 1996) proposed a parallel region-growing implementation for distributed systems that assumes a shared memory with a global address space. The authors of (Montoya et al., 2003) implemented a parallel message passing split-and-merge algorithm, but focused on the problem of load imbalance. Also, (Wassenberg et al., 2009) proposed a graph-based parallel algorithm for RS image segmentation that runs on multicore processors.

In order to handle the growing sizes of RS image data, parallel algorithms typically have to divide an image into tiles and process them independently. The main problem with this approach is how to deal with the segments that touch the borders of the tiles. The work (Michel et al., 2015) proposed a post-processing step for a mean-shift segmentation algorithm that merges neighbor segments on tile borders if their contact surface is large enough. The author of (Tesfamariam, 2011) proposed an edge detection algorithm based on MapReduce that performs edge detection independently and applies a reduction step to merge the results. In (Cao et al., 2014) the authors introduced a parallel k-means clustering algorithm that runs on a cloud environment, their post-processing step, however, is sequential. The authors of (Tilton et al., 2012) proposed a region-growing segmentation algorithm that allows segments with spatially disjoint regions, the algorithm includes a serial processing window artifact elimination step that requires parameters, such as the minimum number of regions and merge threshold to converge. The authors of (Körting et al., 2013) proposed an adaptive tile division approach where the image gradient is used to create tile lines that follow the border of the segments; however, the method might yield erroneous results due to large, highly homogeneous image regions close to the cutting lines. The work (Lassalle et al., 2015) introduced the concept of stability margin for each tile to determine sets of segments that will not be affected by image tile division, their method aims at ensuring equivalent results for a tile-based region-growing segmentation with arbitrary tile sizes in a sequential way.

The authors of (Happ et al., 2016) proposed a distributed region-growing segmentation approach that deals with the border artifacts produced by independent tile processing. Three post-processing strategies were devised to stitch together the segments that touch tile limits. The results produced with an implementation based on the Apache Hadoop framework, showed a considerable reduction of processing times, and segmentation outcomes very similar to those of the sequential execution.

In this work we compare the outcomes of the original implementation of the approach proposed in (Happ et al., 2016), with another implementation, built with Apache Spark. In the next section we briefly describe the approach and its different post-processing strategies, as well as the main characteristics of the Spark and Hadoop distributed computing frameworks.

3. DISTRIBUTED SEGMENTATION APPROACH

A region-growing segmentation procedure iteratively merges neighboring segments until a stopping criteria is reached. The basic idea to distribute the processing of such procedure, is to dived the input image into tiles and process each tile independently. This solution, however, requires some specific mechanism to deal with the segments located at the edges of the tiles, or else the outcome will contain numerous segments with edges affected by artifacts, i.e., straight borders at the tile divisions.

The approach proposed in (Happ et al., 2016) deals with this problem in three steps. Initially, the image is divided into tiles which are distributed to different computing units. Then, each tile is segmented independently. Finally, a post-processing method is used to stitch adjacent segments that touch the edges of the tiles in order to suppress the artifacts generated by the independent segmentations.

It is assumed that the *internal segments*, the ones that do not touch the edges of the tiles, are correctly delineated, that is:

they are not affected by the division of the image. Although this assumption is not necessarily true in all cases, it makes sense primarily because those segments are less subjected to the potential influence of pixels in adjacent tiles. This hypothesis implies a much smaller amount of processing during the stitching process, and allows the segments associated to different tiles to be grouped without critical memory problems.

A particular spatial indexing scheme, with different hierarchical levels, is used to determine the geographical extent of the image tiles, and to label segments. The indexing scheme supports clustering of segments in the post-processing step according to their spatial location. The image division relies on a hierarchical grid of cells, called *geocells*. The top-most geocell layer contains a single geocell, which covers completely the image extent. This cell is subdivided into four geocells in the immediately lower layer. From then on, each geocell is divided recursively into four cells in the next lower layer until a layer with cells of the same size of the desired tile size is reached, thereby forming a quadtree structure.

Three post-processing strategies are defined. The first and simplest has a single step, which allows adjacent segments that touch the same tile borders to be merged, acording to merging rule of the segmentation algorith. This post-processing strategy, called *Simple Post-Processing* (SPP), is the fastest, but still produces a considerable number of artifacts.

The *Hierarchical Post-Processing* (HPP) strategy involves a hierarchical, iterative procedure. The hierarchical geocell levels are used to define a progressive post-processing procedure. Successive steps are performed, until the highest (coarsest) hierarchy geocell level is reached. The procedure for each step is exactly the same, but applied to different collections of segments. At each hierarchical level, the segments that touch the border of the geocells at that level and intersect the same upper level geocell are grouped together. Then, the adjacent segments in each group are allowed to merge. The processing time is longer, but the number of artifacts is reduced as compared to the SPP outcome.

The last and most complete post-processing strategy is called *Hierarchical Post-Processing with Re-segmentation* (HPPR). HPPR is similar to HPP but involves re-segmentation at each hierarchical level. In each iteration step, the groups of tile bordering segments are dissolved back into pixels, and the segmentation procedure is carried out over the full extent of each group. In this way the growth of regions is no longer bounded by the tiles boundaries. This strategy is naturally the slowest, but provides a segmentation outcome without artifacts.

4. DISTRIBUTED COMPUTING FRAMEWORKS

Apache Hadoop comprises a software library and a framework for distributed processing of large data sets across computer clusters. Hadoop can scale from a single machine up to thousands of computing units, in physical or virtual computer clusters. Even though the first official release of Apache Hadoop distribution was deployed in 2011, Hadoop is now the de facto standard in big data applications (Hess, 2016).

The Hadoop framework has three main components: the Hadoop Distributed File System (HDFS), an open source implementation of the Google File System (Ghemawat et al., 2003); the MapReduce API; and a scheduler called YARN (Yet Another Resource Negotiator) (Vavilapalli et al., 2013). In a Hadoop application, HDFS ensures that a sufficient amount of data segments is available, and spread out in the cluster. Through a process called delay scheduling (Zaharia et al., 2010a), YARN tries to maximize data locality in order to reduce network communication. HDFSs architecture follows the master/slave paradigm, the master keeps information about data placement, whereas the slaves store data segments and report their status to the master regularly (Maxdml, 2017). To ensure fault tolerance and availability, data segment sets are replicated in different physical locations, in order to provide resilience to node failures.

Spark belongs to a new generation of Distributed Computing frameworks (Zaharia et al., 2010b), it became an Apache project in 2013. Spark is compatible with Hadoops modules, such as YARN and HDFS, but it also has a standalone mode. The key motivation behind the Spark project was to enhance the performance of iterative workloads through in-memory computations. Due to the numerous disk access required to process an application, Hadoop is quite inefficient for such workloads (Maxdml, 2017).

Such in-memory computations rely on a data structure called Resilient Distributed Dataset (RDD), which are fault-tolerant collections of elements that can be operated on in parallel, and can be used to cache a dataset in memory across operations. RDDs can reference a dataset in an external storage system, such as a HDFS, and can be created from any storage source supported by Hadoop. RDDs are lazily computed, in the sense that sequence of transformations on it will only be process when the associate data needs to be collected. Moreover, if any data partition of an RDD is lost due to physical errors, Sparks cache can automatically be recomputed by re-executing its respective sequence of transformations (Hess, 2016).

5. EXPERIMENTAL ANALYSIS

In order to evaluate the alternative implementations of the distributed segmentation approach, we performed experiments using a WorldView-2 scene, acquired in 2012. The scene covers urban and rural areas of the São José dos Campos municipality, in São Paulo state, Brazil.

Figure 1. WorldView-2 image used in the experiments. (In this figure the image was rotated by 90 degrees.)

The full image (Figure 1), hereinafter called 16K image, is a pansharpened, 0.5-m spatial resolution image, with

11370×16000 pixels, and three bands (red, green, and blue). In the experiments we also used subsets of this image with 5685×8000 and 2844×4000 pixels, denoted in the following as 8K and 4K images. The subsets were taken from the center of the full image. Tile size was set to 1024×1024 pixels.

The region-growing image segmentation algorithm built into the implementations of the distributed segmentation approach is the one proposed in (Baatz , Schäpe, 2000). This choice was due to its complexity and popularity among the remote sensing community. Its parameter values were kept the same in every run: scale = 40; color weight = 0.84; compactness = 0.8; bands weights = 1, 1, 1; merging heuristic = local mutual best fitting.

The experiments were carried out in a physical cluster composed of 10 nodes. Every cluster node has two Intel R Xeon-E5345, with 4 cores each, operating at 2.33 GHz, with a 64-bit architecture, 8 GB DDR2 667MT/s RAM and a 146 GB hard disk, with 10k RPM. We used version 2.6 of Hadoop and 2.1 of Spark.

Figures 2 to 4 show the processing times of the segmentations performed with the alternative implementations, for all images and post-processing strategies. Running on clusters with 1, 3, 6, and 9 nodes, for the 4K and 8K images, and with 3, 5, 7 and 9 nodes, for the 16K image. In all cases one more machine was reserved for the YARN Resource Manager. We decided to start with a three machine cluster in the case of the 16K image, because Spark's Driver Program was consuming all resources from one cluster node, which was then not participating in the segmentation task. In the figure legends the S suffix identifies the Spark implementation and the H suffix, the Hadoop implementation.

Speedups were computed as a ratio, considering the execution time obtained with the smallest number of nodes, for the same framework and post-processing strategy.

The figures show that the Spark implementation consistently, and in most cases significantly, outperforms the Hadoop counterpart in terms of processing time, for all cluster configurations and post-processing strategies.

Moreover, the speedups associated with Spark were consistently higher, but some times similar to those obtained with Hadoop. It is noteworthy that, especially for the hierarchical post-processing strategies, speedups tend to decrease in rate, as more machines are used. This has to do with the fact that in those strategies, as processing reaches the higher geocell hierarchy levels, the number of tasks decrease, up to a point that adding more machines will not result in a linear decrease in processing time. Anyhow, we believe these results confirm the scalability potential of the alternative frameworks in the context of tile-based region-growing segmentation.

Framework/Strategy	SPP	HPP	HPPR
Hadoop×Spark (9 nodes)	0.992	0.991	1.000
Hadoop (1 node×9 nodes)	0.994	0.999	1.000
Spark (1 node×9 nodes)	0.988	1.000	1.000

Table 1. Comparison of the segmentation outcomes according to Hoover metric.

As for the stability of the two implementations, we compared segmentation outcomes using the Hoover metrics (Hoover et al., 1996). In Table 1 we show the values associated with the

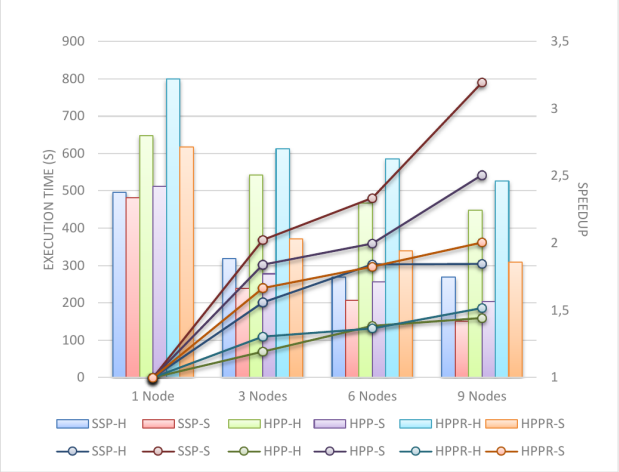

Figure 2. Segmentation of the 4K image.

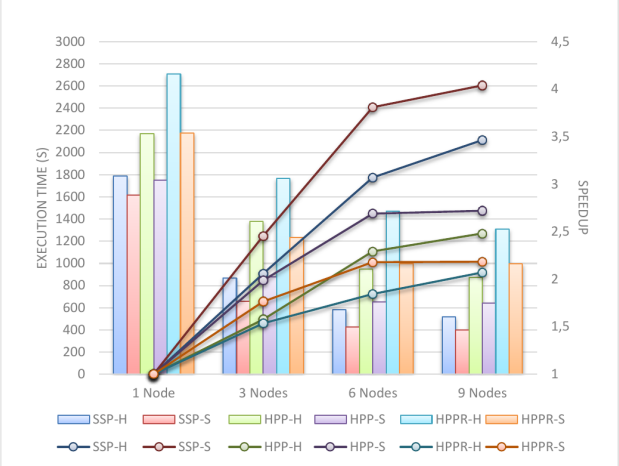

Figure 3. Segmentation of the 8K image.

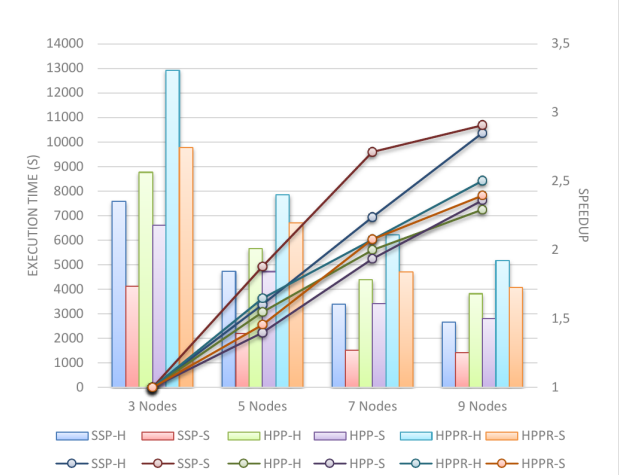

Figure 4. Segmentation of the 16K image.

segmentation of the 4K image, for the three post-processing strategies. We first compared the segmentation produced with the Hadoop and the Spark implementations running on nine nodes. Then we compared the outcomes of each implementation running on one and nine nodes. Recalling that

the value 1.0 (one) represents a perfect match, we can conclude that the two implementations generated very similar results. In the case of the HPPR strategy, all results are identical. The discrepancies noted for the other strategies can be explained by the irregular latencies associated to cluster computing, which can interfere in the order of selecting segments for merging. Anyhow, a Hoover value of the order of 0.995 indicates that discrepancies where found in approximately 60 segments out of 13,400, what represents an insignificant difference.

6. CONCLUSIONS

In this work, we developed and evaluated a new implementation of the distributed segmentation approach proposed in (Happ et al., 2016), built on top of the Spark distributed computing framework, and compared its performance with that of the original implementation, built using the Hadoop framework. In this investigation, we made a number of experiments processing RS images of different sizes, using clusters with varying number of processing units. In addition to comparing computational performances in terms of processing times, we accessed the discrepancies among the various segmentation outcomes.

With respect to computational performance, the experiments show that the Spark implementation consistently outperformed the Hadoop counterpart, for all cluster configurations and for all post-processing strategies. Additionally, in most cases the improvement brought by using Spark was very significant in terms of processing times. The experiments also show the stability of the distributed segmentation approach, in the sense that it produces very similar results, if not identical, regardless of the distributed framework and of the cluster configuration.

As a continuation of this research, we plan to run experiments on larger clusters, provided by cloud computing infrastructure services, in order to further investigate the scalability potential and limitations of the general approach and its particular implementations. We also want to investigate the stability of the segmentation outcomes with respect to varying image tile sizes.

ACKNOWLEDGEMENTS

This work is supported by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and FAPERJ (Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro).

REFERENCES

Apache Hadoop Development Team, 2019. Apache hadoop. Apache Software Foundation. hadoop.apache.org (20 February 2019).

Apache Spark Development Team, 2019. Apache spark. Apache Software Foundation. spark.apache.org (20 February 2019).

Baatz, M., Schäpe, A., 2000. Multiresolution segmentation: an optimization approach for high quality multi-scale image segmentation. *Angewandte Geographische Informationsverarbeitung XII, Heidelberg*.

Barder, D.A., Jaja, J., Harwood, D., Davis, L.S., 1996. Parallel algorithms for image enhancement and segmentation by region growing with an experimental study. *10th Int. Symp. Parallel Process. Honolulu, HI, USA*.

Blumenfeld, J., 2019. Getting petabytes to people: How eosdis facilitates earth observing data discovery and use. NASA EODIS EARTHDATA.

Câmara, G., Souza, R.C.M., Freitas, U.M., Garrido, J., 1996. Spring: Integrating remote sensing and GIS by object-oriented data modelling. *Computers Graphics*, 20(3), 395–403.

Cao, X., Li, Q., Du, X., Zhang, M., Zheng, X., 2014. Exploring effect of segmentation scale on orient-based crop identification using hj ccd data in northeast china. *IOP Conference Series: Earth and Environmental Science, Conference 1*, 17.

Castriotta, A.G, Knowelden, R., 2018. Sentinel data access annual report 2017. COPERNICUS-SERCO Report 17-0186.

Dean, J., Ghemawa, S., 2008. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1), 107–113.

Ghemawat, S., H., Gobio, Leung, S.-T., 2003. The google file system. *ACM SIGOPS operating systems review*, 1, 29–43.

Happ, P.N., Costa, G.A.O.P., Bentes, C., Feitosa, RQ., Ferreira, R.S., Farias, R., 2016. A cloud computing strategy for region-growing segmentation. *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, 9, 5294-5303. doi: 10.1109/jstars.2016.2591519.

Hess, K., 2016. Hadoop vs. spark: The new age of big data. Datamation.

Hoover, A., Jean-Baptiste, G., Jiang, X., Flynn, P.J., Bunke, H., Goldgof, D.B., Bowyer, K., Eggert, D.W., A., Fitzgibbon, Fisher, R.B., 1996. An experimental comparison of range image segmentation algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(7), 673-689.

Körting, T.S., Castejon, E.F., Fonseca, L.M.G., 2013. *The divide and segment method for parallel image segmentation*. Springer, New York, 504–515.

Lassalle, P., Inglada, J., Michel, J., Grizonnet, M., Malik, J., 2015. A scalable tile-based framework for region-merging segmentation. *IEEE Trans.Geosci. Remote Sens.*, 53(10), 5473-5485.

Lee, J. G., Kang, M., 2017. Geospatial big data: challenges and opportunities. *Big Data Research*, 2, 74–81. doi:10.1016/j.bdr.2015.01.003.

Maxdml, 2017. An overview of distributed computing frameworks.

Michel, J., Youssefi, D., Grizonnet, M., 2015. Stable Mean-Shift Algorithm and Its Application to the Segmentation of Arbitrarily Large Remote Sensing Images. *IEEE Trans.Geosci. Remote Sens.*, 53, 952–964.

Montoya, M. D. G., Gil, C., Garca, I., 2003. The load unbalancing problem for region growing image segmentation algorithms. *J. Parallel Distrib. Comput.*, 63(4), 387-395.

Saha, B., Shah, H., Seth, S., Vijayaraghavan, G., Murthy, M., C., Curino., 2015. A unifying framework for modeling and building data processing applications. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*, 1357–1369.

Tesfamariam, E.B., 2011. *Distributed processing of large remote sensing images using mapreduce A case of edge detection*. Inst. for Geoinformatics, University of Muenster, Universitat Jaume I, and Universidade Nova de Lisboa, Muenster.

Tilton, J.C., Tarabalka, Y., Montesano, P.M., Gofman, E., 2012. Best merge region-growing segmentation with integrated nonadjacent region object aggregation. *IEEE Trans.Geosci. Remote Sens.*, 50(11), 4454-4467.

Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., S., Seth, 2013. Apache hadoop yarn: Yet another resource negotiator. *Proceedings of the 4th annual Symposium on Cloud Computing*.

Wassenberg, J., Middelmann, W., Sanders, P., 2009. *An efficient parallel algorithm for graph-based image segmentation*. 5702, Springer, Berlin, 1003–1010.

Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., I., Stoica, 2010a. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. *Proceedings of the 5th European conference on Computer systems*, 265–278.

Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., I., Stoica, 2010b. Spark: cluster computing with working sets. *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*.